LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SOLO PROGRAMADORES

AÑO VI. NÚMERO 57

UNA PUBLICACIÓN DE: TOWER COMMUNICATIONS

975 Ptas. • 5,86 € (IVA incluido)

SONIDO Y VÍDEO CON JAVA (II)
Java Sound y los fundamentos de Java Media Player

APLICACIONES CON VIDEO CONFERENCIA (III)
Programación de NetMeeting con C++

APLICACIONES IIS EN VISUAL BASIC (Y II)
Cómo programar servidores Internet

GLIDE (Y VI)
Estados, formatos de texturas y modo



Captura de vídeo con DELPHI



BASES DE DATOS CON ORACLE Y SQL SERVER
Desarrollo de aplicaciones cliente/servidor

DIRECTX 6.1 (II)
Captura de sonido con la API DirectSound

XML (II)
Aspectos básicos del modelo DOM y su utilización

BORLAND C++ BUILDER 4
Potente y Visual: una mezcla explosiva



CD CONTENIDO

• BORLAND C++ BUILDER 4 • APACHE HTTP SERVER 1.3.6 • KRYPTOS DEVELOPER 1.1
• ULEAD ANIMATION.APPLET 1.0 • YES2K 2.05 • TCL/TK 8.0.5 • DOMINHTML 4.0

SÓLO PROGRAMADORES

Número 57
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS
Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Subdirector
Oscar Rodríguez Fernández
oscarrf@towercom.es
Coordinador Técnico
Eduardo De Riquer Frutos
eriquer@towercom.es
Coordinadora de Redacción
Erika de la Riva Arnáiz
eriva@towercom.es
Colaboradores
Constantino Sánchez, Juan Luis Ceada, Jorge Delgado, Javier Sanz, Adolfo Aladro, Javier Toledo, Victoria Rus, Esteban Amado, Jordi Agost
Maquetación y Tratamiento de Imagen
Ana Isabel Madero Bocos

Publicidad
Inmaculada Romera (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Alicia Zazo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Javier Amado (Jefe)
jamado@towercom.es
Servicio Técnico
Jaime Bort
jaimeb@towercom.es

Preimpresión
IndesColor
Impresión
Gráficas Muriel
Distribución
SGEL
Distribución en Argentina / Chile / Colombia / México / Venezuela
Capital: Huesca y Sanabria
Interior: D.G.P.

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es
Distribución
Almiro Sanguino

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 31-8-99

Editorial

Es el momento de los humildes mortales

Cada vez somos más los incondicionales reunidos entorno a los innumerables apartados que abarca la informática, muchos de los cuales resultaban inviables técnicamente hace menos de 15 años. La verdad es que sólo la tecnología frenaba su desarrollo, puesto que muchos objetos ya formaban parte de la bolsa de viaje de "héroes" cuya única credibilidad residía en su equipaje. Pues bien, ha llegado el momento de los humildes mortales, ya que maravillas como la videoconferencia, la captura y edición de vídeo o la posibilidad de ver la televisión en el *PC* son una realidad actual. Y lo mejor de todo es que su asequible precio se adapta de forma bastante precisa a todas las connotaciones de la palabra humilde, haciendo que se conviertan en otro periférico orientado al mercado de consumo.

Ante esta halagüeña situación, y ya desde el prisma del programador, podemos deducir que hoy en día el *hardware* posee un relevancia cada vez mayor. No me refiero a su presencia en grandes compañías y organismos oficiales, sino al que se encuentra con gran asiduidad en el mercado de consumo. Sí, porque una vez alcanzada esta cuota, la necesidad de conocer sus características técnicas es vital para que sigamos progresando en el campo de la programación, donde de nuevo se abre la puerta con trabajo. No debemos olvidar que el chollo del año 2000 tendrá que terminar en algún momento y no durará eternamente (es posible que en España sí, claro).

Si en números anteriores abordamos el tema de la videoconferencia mediante *NetMeeting*, ahora le ha llegado el turno a la captura de vídeo. El tema de portada recoge la exposición anterior y mediante un entorno *RAD* (*Delphi* en este caso), explica cómo crear un componente que permite trabajar de forma transparente con la capturadora. Aunque se explican de forma detallada las características técnicas necesarias para la creación del componente, la posibilidad de utilizarlo en nuestras aplicaciones sin entrar en interioridades también es una realidad. Por cierto y hablando de entornos *RAD*, este mes comentamos la aparición estelar de *C++ Builder 4*, una herramienta que aprovecha toda la potencia del lenguaje *C*, honrando de forma notable la palabra Visual. Para no abandonar el tema de las capturas, también merece atención el tema dedicado al sonido y a su manipulación mediante la *API DirectSound* del nuevo *DirectX 6*. En este caso y con el fin de aprender dos aspectos conjuntos en cada artículo, el lenguaje utilizado será *C++*.

Como no podía ser de otra manera, *Java* sigue entre nosotros y para no ser menos, el artículo trata los dos temas anteriores, pero a través de las *APIs Java Sound* y *Java Media Player*. Como punto final no podía faltar, el de momento último gran impulsor de la Informática, *Internet*. Para ser más concretos el desarrollo de servidores *IIS* con *Visual Basic* y continuaremos con la serie dedicada al lenguaje *XML*. Esperamos que todas estas posibilidades sean útiles para todos los lectores de nuestra revista.

SÓLO PROGRAMADORES

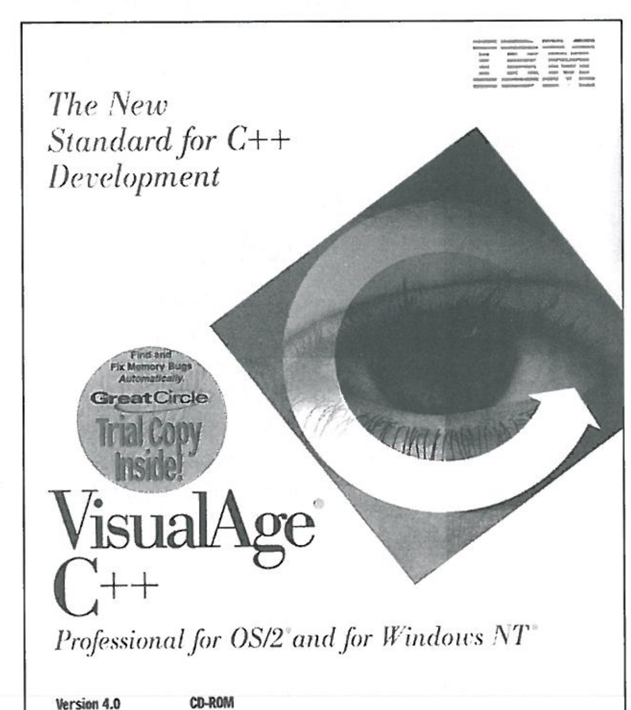


IBM SE UNE A LAS ÚLTIMAS ACTUALIZACIONES DE COMPILADORES DE C++

El gigante azul (*IBM*) acaba de lanzar al mercado la versión 4.0 del compilador de *C++* dentro de la familia de productos *VisualAge*. Este compilador permite desarrollar aplicaciones multiplataforma, aunque solamente se podrá instalar en equipos que tengan el Sistema Operativo *Windows NT* o *OS/2*, permite la distribución de las aplicaciones generadas en sistemas *Windows* 95 ó 98.

Para poder disfrutar de las nuevas características de este entorno de desarrollo, han de cumplirse los siguientes requisitos: un procesador *Intel Pentium* 166 o superior, una *SVGA* de 800x600, aunque se recomienda que sea capaz de alcanzar una resolución de 1024x768, un lector de CD-ROM, ratón, y dependiendo del sistema operativo donde se vaya a instalar, se necesitarán 630 *MB* para una instalación completa sobre *OS/2* con *HPFS*, y 570 *MB* en instalaciones sobre *Windows NT* con *NTFS*. La instalación mínima requiere 45 *MB*.

Como requisitos de software, *IBM* nos advierte que para poder instalar este producto se necesita *OS/2 Warp* versión 4 o *Windows NT* 4.0; pero además se necesita poseer un *browser HTML*, que permita acceder al nuevo sistema de ayuda; el cual ha de ser como mínimo *Netscape Navigator* 2.02 para OS/2, mientras que para *Windows NT* como mínimo se ha de poseer *Netscape* 3.51 o *Explorer* 4.0.



Dentro de las nuevas características, podemos resaltar:

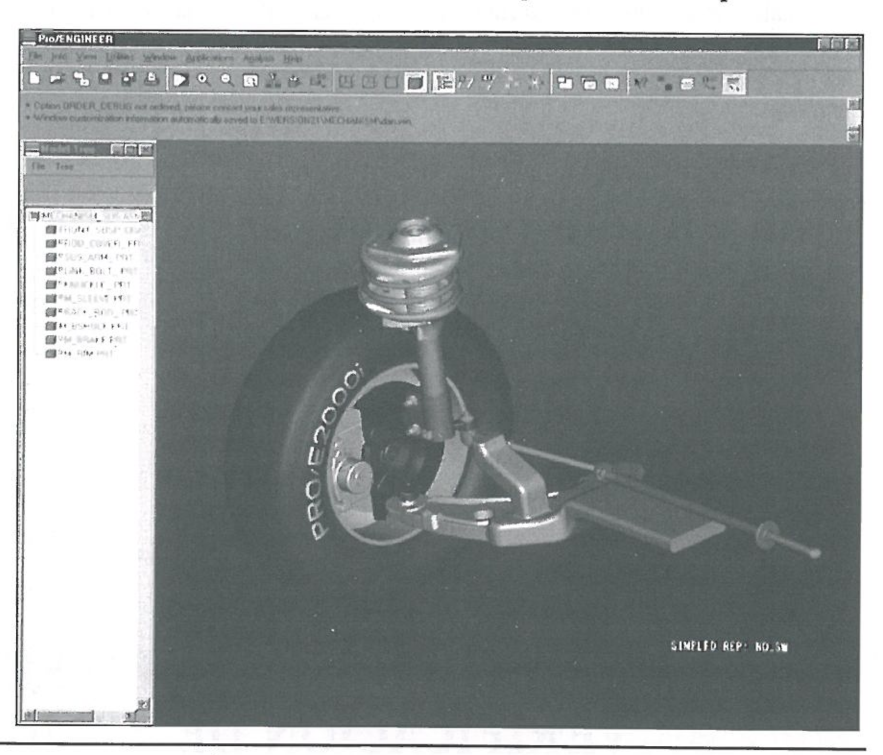
- Mejora del entorno *IDE*: se ha organizado el entorno de desarrollo para mejorar la forma de desarrollar aplicaciones, encontramos:
  - *WorkBook*: desde la cual se permite examinar y establecer las opciones de control del *IDE*.
  - *Host*: Sección que muestra los ficheros incluidos en el proyecto.
  - *Proyect*: Desde esta sección se tiene acceso al código fuente.
  - *Configuration*: Desde aquí se establecerán las opciones de compatibilidad y optimización.
- Nuevas posibilidades del lenguaje: La nueva versión de *VisualAge C++* soporta programación *orderless* (sin orden). Por esto la nueva tecnología que *IBM* ha aplicado permite olvidarse de la necesidad de preocuparse del orden en que se declaran las clases.
- Por otra parte, *IBM* proporciona su *Open Library Class* (librería de clases) diseñadas para aumentar la facilidad de desarrollo reduciendo así el esfuerzo de los programadores. A partir de estas clases, se podrá generar interfaces de usuarios, accesos a bases de datos, posibilidad de colecciones, de tipos de datos y nuevas excepciones.
- Los nuevos asistentes, *SmartGuides*, simplifican las tareas que podrían resultar complicadas; *Proyect SmartGuide* proporciona ayuda a la hora de establecer las características del nuevo proyecto, *Target SmartGuide* permite añadir un nuevo tipo de destino al proyecto, ó el *Parts SmartGuide* que facilita la creación de las partes visuales.

Se puede ampliar información en en la página: **http://www.software.ibm.com/visualage_c++/**

PARAMETRIC TECHNOLOGY PRESENTA PRO/ENGINEER 2000i

La próxima generación de soluciones informáticas de Parametric viene marcada por la nueva tecnología de modelado por comportamiento.

Hablamos de un novedoso conjunto de herramientas de software para desarrollo y diseño de productos, que incorpora el modelado por comportamiento, una revolucionaria tecnología que permite la excelencia en el diseño gracias a las herramientas orientadas a objetos. Esta versión, Pro/ENGINEER 2000i, es la nueva piedra angular de las recientes Series-i de soluciones de software para el desarrollo de productos. Incluye más de 500 mejoras, presenta también nuevas funcionalidades y aplicaciones para grandes conjuntos, características de adaptación de proceso, animación de diseño y herramientas de producción de fabricación. Esta versión incorpora, además, mejoras para compartir a través de Internet, información sobre la ingeniería y proporciona avances significativos gracias al interfaz gráfico de usuario. Se puede ampliar la información sobre éste producto en **www.ptc.com**



LEXMARK AÑADE SOPORTE LINUX A MARKVISION PARA REDES UNIX

MarkVision, el software de impresión para redes UNIX de la compañía Lexmark soportará a partir de ahora Linux. Con esto los usuarios de Linux Red Hat 5.2 obtienen potentes funciones de gestión para impresoras como por ejemplo el control remoto en tiempo real, la detección de impresoras nuevas o la fácil y rápida configuración de cola e impresora. Por supuesto, todo ello conlleva una importante reducción del tiempo que hay que invertir en la instalación y configuración de este hardware.

Además de Linux Red Hat 5.2, también admite Sun Solaris 7 y SCO UnixWare. El precio estimado del software es de 43.700 ptas. Más información en **www.lexmark.com**

MEJORADA LA OFERTA DE BUSINESS INTELLIGENCE PARA SEAGATE HOLOS 7

La empresa Seagate anuncia la disponibilidad de aplicaciones analíticas para Seagate Holos 7. Estas plantillas, que estarán disponibles en el segundo trimestre de 1999, se basan en aplicaciones para Presupuestos, Segmentación de clientes y Balanced Scorecard.

Seagate Holos proporciona un entorno de desarrollo escalable y flexible para ofrecer aplicaciones analíticas personalizadas que aprovechan los, cada vez mayores, volúmenes de datos corporativos; y proporcionar a los usuarios un análisis de la información significativa, facilitando de este modo la toma de decisiones en la empresa. Para obtener más información sobre este producto consultar en **www.seagatesoftware.com**

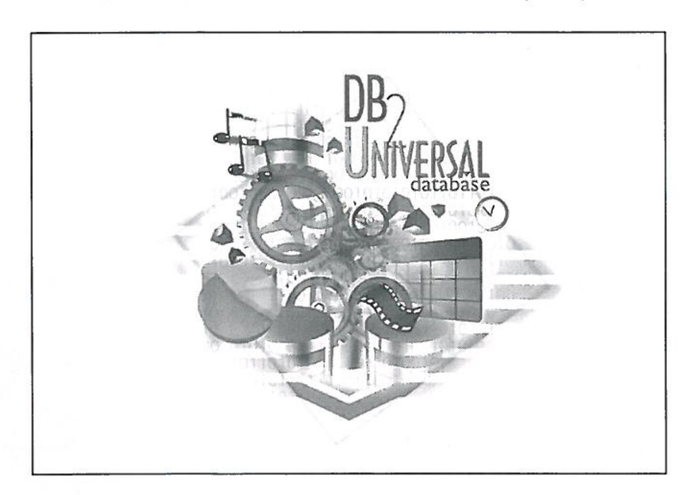
IBM ANUNCIA DB2 PARA AS/400

IBM presenta la versión para AS/400 de DB2 Universal Database, el primer sistema de gestión de bases de datos relacionales preparado para Internet con la potencia suficiente para satisfacer las demandas de las grandes organizaciones y flexible para operar en las pequeñas y medianas empresas.

Con esto se conseguirá que más de 20 millones de usuarios de AS/400 se beneficien de las ventajas que este sistema de gestión ofrece, entre los que figuran la capacidad para gestionar y analizar contenido multimedia, ofrecer soporte para aplicaciones Business Intelligence complejas, así como integrar las aplicaciones existentes en la empresa con las e-business.

Estamos hablando de una base de datos escalable y preparada para la Web que gestiona un tipo de datos complejos y que incluye soporte para Java mejorando la integración con IBM WebSphere Application Server.

Además IBM ha suscrito un acuerdo con ShowCase Corporation para suministrar nuevas herramientas de proceso analítico on-line (OLAP) y data warehouse. Si deseas ampliar la información sobre este producto puedes consultar en la dirección de Internet **www.software.ibm.com/data/db2**



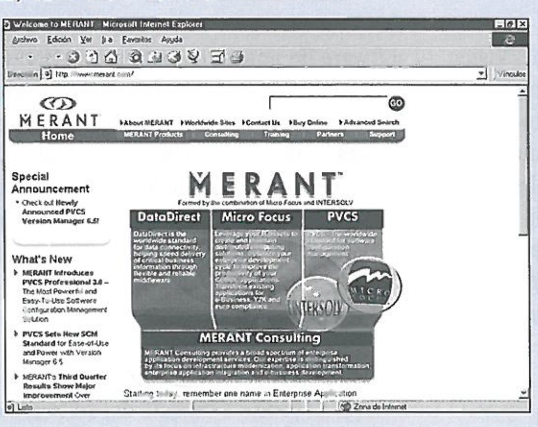
MERANT PRESENTA UN SERVICIO DE VERIFICACIÓN DE APLICACIONES

Merant Consulting ha dado a conocer el servicio de Verificación de aplicaciones, su última solución para comprobar si los sistemas de información se adaptan a los requisitos de año 2000.

La Verificación de aplicaciones ayuda a las empresas a determinar si el código fuente de sus programas está preparado para el cambio del milenio. Mediante unas herramientas especiales de Merant Micro Focus, se identifican rápidamente errores, permitiendo a los consultores analizar el código COBOL corregido y confirmar su nivel de conformidad y adecuación al año 2000.

Este servicio de Verificación de aplicaciones suministra un informe con una lista detallada de errores o problemas potenciales detectados en las aplicaciones examinadas. El informe de este servicio, además, proporciona un sistema independiente de certificación y comprobación del estado del código revisado, contiene los detalles del total de programas donde se han detectado los fallos y una lista de las líneas de código que precisan corrección.

La utilización de este sistema es especialmente útil para aquellas organizaciones que han subcontratado los proyectos de adaptación al 2000, que han utilizado herramientas de búsqueda y comparación automática o han optado por una revisión manual del código. Si precisas más información sobre la compañía y sus productos consulta en la Web en **www.merant.com**

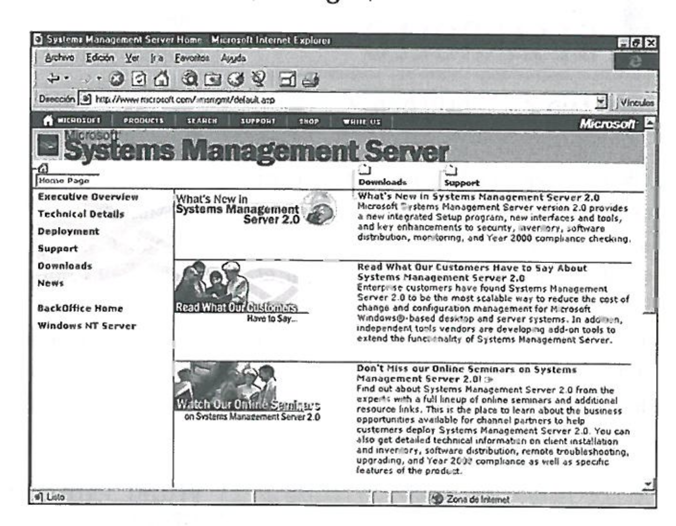


MICROSOFT PRESENTA SYSTEMS MANAGEMENT SERVER

La nueva versión de Systems Management Server está ya disponible según un anuncio hecho por la compañía Microsoft. Se trata de un software que facilita a los usuarios la gestión centralizada de todos sus sistemas de sobremesa y servidor basados en el sistema operativo Windows. Este sistema puede ser utilizado para reducir el coste derivado de la gestión en la configuración y el cambio de componentes software para los servidores y sistemas de sobremesa basados en Windows.

SMS ayuda tanto a las PYMES como a las grandes empresas a responder a este desafío, al permitirles realizar un inventario de los recursos de hardware y software disponibles, distribuir e instalar software de forma remota así como el soporte de usuarios desde una localización central.

Estas capacidades también ayudan a los administradores a identificar y resolver los problemas derivados del Efecto 2000 en sus entornos basados en Windows. El producto incluye diversas herramientas como por ejemplo de planificación, de implantación y de diagnóstico. Si deseas obtener información adicional acerca de este producto puedes consultar en la siguiente dirección en Internet de la compañía **www.microsoft.com/smsmgmt/**



PLATINUM HACE PÚBLICA LA LLAVE DE LA VISUALIZACIÓN 3D EN INTERNET

Platinum ha anunciado la intención de ceder al público el código fuente de sus soluciones de visualización en 3D para Internet a través del Web3D Consortium. Estos productos basados en el estándar VRML, incluye la herramienta de autor Cosmo World 3D y el plug-in para visualizar contenido 3D en Internet, Cosmo Player.

La estrategia de la compañía está diseñada para ampliar el mercado de visualización 3D en Internet incrementando de forma significativa el número de desarrolladores que producen software y contenidos basados en el código fuente de Platinum, además la empresa también suministrará potentes soluciones corporativas con mejoras visuales que aprovechan la tecnología 3D basada en estándares abiertos.

Por otro lado Platinum, aunque suministre el código al Web3D Consortium, conservará en todo momento los derechos de propiedad intelectual y la patente de la tecnología.

Web3D será el organismo que gestione la distribución de licencias de código, también se espera que este código fuente sirva de base para la implantación del X3D, un estándar de próxima generación. Amplía esta información en **www.web3d.org** o en **www.vrml.org**

HERRAMIENTAS DE PROGRAMACIÓN



ENTORNOS DE DESARROLLO

BORLAND C++ BUILDER 4

Última versión de la herramienta visual estándar para el desarrollo de aplicaciones C++. Entorno de desarrollo totalmente visual que permite programar el interfaz gráfico mediante el arrastre de los componentes que Borland ofrece a través de sus VCL para posteriormente asociar el código que ha de ejecutarse como respuesta de los eventos.

NOTA: Para poder utilizar esta versión de evaluación es imprescindible registrarse. Una vez relleno el cuestionario del registro que acompaña al programa, éste se envía automáticamente por Internet y la casa Borland devuelve a través de E-mail el número de serie necesario.

WORLD STUDIO 98 1.20

Entorno de desarrollo VRML para la creación de mundos virtuales 3D para la Web. Es una solución intuitiva que ofrece posibilidades drag-and-drop para crear ficheros VRML 2.0.

Basta con arrastrar un objeto y situarlo dentro del entorno. Después basta con configurar las características del objeto, su movimiento, los efectos de sonido, etc.

DEV-C++ 2.0

Entorno de desarrollo y compilador de C++ freeware. Consiste en un editor multiventana que integra un compilador de acceso rápido que permite realizar el proceso y su ejecución. El editor ofrece capacidades para el reconocimiento de sintaxis. También incluye un debugger y un creador de instalaciones. Una buena opción gratuita.

TCL/TK 8.0.5

Se basa en el Tool Command Language de Sun e incluye un interfaz gráfico. TCL (Tool Command Language) es un lenguaje de scripting y TK es el interfaz gráfico basado en TCL para facilitar su uso. Gracias a este paquete se pueden desarrollar aplicaciones Windows usando una mínima cantidad de código.

VISUAL PROGRAMMING ARMOURY 1.10

Es un IDE multipropósito diseñado para programadores C++ y Java. Proporciona al desarrollador una interfaz muy bien diseñada que permite gestionar grandes proyectos de programación de manera muy sencillo.

Ofrece diseño visual de formularios, mapas y bases de datos. Es freeware.

WINEDIT 99D

Winedit se trata de un editor de textos. Potente y flexible editor de textos especialmente diseñado para desarrolladores.

Incluye opciones tan interesantes como por ejemplo: buscar y reemplazar, tamaño ajustable de fuentes, cabeceras, numeración de páginas, impresión por doble cara, apertura de múltiples ficheros, deshacer y rehacer, coloreado de código y sintaxis, carga rápida de ficheros y scrolling, plantillas...

LENGUAJES

VISUAL BASIC

CODESMART 3.2

Añadido de Visual Basic que incluye gran número de nuevas funcionalidades. Entre otras características el programa implementa: items de código coloreados, añade números de línea, una base de datos actualizable de código, un sistema de creación de macros, etc.

VBGUARD 2.2

Sistema para el control y recuperación de errores de aplicaciones Visual Basic. Intercepta errores en las runtime y programas tras el cuelgue o el fallo de la aplicación. Puede mostrar argumentos de procedimientos, variables (incluyendo objetos y arrays). Otras característica son: notificación remota, editor integrado, informe detallado y listado HTML.

JAVA

COOLBE (LIGHT EDITION) 1.3

Sistema para la creación sencilla de slideshows y banners en Java. Cada applet puede incluir más de 200 temas distintos que pueden estar compuestos de links estáticos o de imágenes animadas. Se puede utilizar junto con cualquier editor de textos y tan sólo basta con abrir una de las plantillas y personalizar los parámetros y la configuración. Es freeware.

ULEAD ANIMATION.APPLET 1.0

Potente sistema de desarrollo rápido de animaciones basadas en Java.

Se pueden crear bandas informativas, presentaciones, banners, animaciones, etc. sin utilización alguna de código Java.

HTML

BUTTONWIZ 5.5

Generación automática de botones Web. Dispone de más de 200 botones predefinidos que pueden ser modificados con facilidad. Puede generar imágenes que pueden ser copiadas en Portapapeles y ser salvadas en formatos PNG, BMP, TIF, JPG, PCX y TGA.



DOMINHTML 4.0

Editor Web gratuito con un diseño muy interesante. Entre otras características DominHTML incluye: tags coloreados, plantillas definidas por el usuario, librería de código más usado, inserción rápida de código, barra de herramientas configurable, etc.

NOTETAB PRO 4.6

Profesional, avanzado y flexible gestor/editor HTML. Dispone de una opción muy innovadora denominada Clipbook que permite la grabación de elementos Web en una librería para su almacenaje y posterior reutilización.

OTROS

ABAKUS VCL 1.30

Es un interesante set gratuito de componentes adicionales para Delphi. El paquete incluye: medidores, barras, indicadores digitales, displays, dials y componentes de todo tipo que tienen que ver con relojes y el tiempo.

KRYPTOS DEVELOPER 1.1

Proporciona herramientas de seguridad de primera clase en desarrollos comerciales y corporativos Delphi 3 ó 4. Mediante un sencillo conjunto de controles OCX ó VCL, simplemente arrastrando y soltando en su entorno de desarrollo Kryptos Developer permite incorporar encriptación por cifrado, funciones Hash, generadores de números pseudoaleatorios, codificación de datos.

DYNAMICUBE 2.0.3.7

Se trata de un control Active X abierto muy dirigido a la creación de aplicaciones cliente/servidor en Internet que precisen la gestión de bases de datos. Ofrece soporte también a usuarios de SQL para acceder a recursos de datos ODBC, DAO, RDO y BDE.

HERRAMIENTAS Y UTILIDADES

INTERNET EXPLORER 5.0

La última versión en castellano de la suite Internet de Microsoft. Incluye un proceso de instalación más detallado y personalizado, un Historial mejorado, una búsqueda en la Web mucho más potente e incluso se ha mejorado la organización de los Favoritos con un acceso más directo y una mejor estructuración. El fichero de instalación es IE5setup.exe

HAPPY99CLEANER 2.1

Utilidad para la eliminación rápida del virus Happy 99. Es una interesante opción para desinfectar los equipos atacados por uno de los virus más populares de los últimos meses. Se trata de un virus que se enmascara tras un fichero denominado Happy99.exe adjunto a un mensaje de correo electrónico. Lo elimina eficiente. Es freeware.

ICOHOLICS ANONYMOUS 4.B

Impresionante colección de más de 2.100 iconos. Todos los iconos están diseñados en 32 x 32 píxeles y 16 e. Incluye categorías como: flechas, botones, perros, comida, hardware, etiquetas, música, smileys, símbolos, dibujos.

REGKEY 3.2

Sistema de control para el registro de nuestras aplicaciones. Ideal para el control del registro tras la creación de aplicaciones shareware o de tiempo limitado. Es posible utilizarlo en aplicaciones Visual C/C++, Visual Basic o Delphi.

WINHEX 8.23

Editor hexadecimal que precisa mínimos requerimientos. Ideal para aquellos desarrolladores que precisen de un editor rápido, que necesite poca memoria y de uso muy sencillo. Permite concatenar, analizar, comparar y convertir ficheros.

YES2K 2.05

Test de compatibilidad del año 2000 en equipos monopuestos o redes. Detecta problemas de compatibilidad en CMOS, BIOS y fecha del RTC. Cuando se ejecuta en una LAN el programa testea las máquinas y ofrece un informe de resultados a la consola central de gestión. El administrador puede realizar los tests de compatibilidad del hardware de red automáticamente

REDES

LOCALES

MAILBEAMER 3.22

Excelente servidor SMTP y POP3 para la creación de oficinas postales en la LAN. MailBeamer se integra en MSMail para permitir a los clientes acceder inmediatamente a los mensajes sin la instalación de ningún otro programa en los clientes. El programa soporta MIME y UUENCODE.

AGGRESSOR TRIAL 1.0

Programa monitorizador y testeador de la vulnerabilidad de una red.

The Aggressor dispone de un módulo avanzado para el traceado y la monitorización del tráfico de una red local. Permite controlar todas las conexiones y los envíos y recepciones de paquetes.

DISTRIBUIDAS

APACHE HTTP SERVER 1.3.6

Servidor HTTP muy popular, rápido, y potente. Se calcula que Apache es el servidor HTTP utilizado en más de un 50% del total de los Web sites de Internet. Ofrece rapidez, estabilidad y potencia además de gran número de opciones para la personalización y configuración.

SERVERPROTECT 4.61

Solución de red para el análisis centralizado contra virus. Permite obtener una protección completa de la LAN contra cualquier tipo de virus. Su opción ServerProtect ofrece conexiones remotas seguras, detalles sobre la actividad del servidor, instalación y gestión bajo varios dominios, escaneo automático y un método de actualización sencillo.

OTROS

ASP BEAUTIFY 1.0

Utilidad para limpiar de código extraño los ficheros ASP. Programa que puede trabajar tanto con código VBScript como con HTML para eliminar espacios y tabulaciones de manera automática. Se puede utilizar para optimizar código existente en el Portapapeles. Precisa de mínimos requerimientos.

REMOTELYANYWHERE 2.32

Programa de gestión y administración remota de redes basada en Web. Permite arrancar y finalizar servicios de NT, iniciar y finalizar procesos, reiniciar la máquina, administración de usuarios, gestión de ficheros, editar el Registro del sistema y tomar control del teclado y el ratón del sistema. Todo se hace a través de un navegador Web.

DOCUMENTACIÓN /TUTORIALES

BEGINNING VISUAL BASIC 1.0

Guía práctica para el aprendizaje de Visual Basic. Curso diseñado para principiantes que introduce al usuario en los elementos del lenguaje y del entorno de programación. Está desarrollado paso a paso. Incluye cinco lecciones: "Introducing Visual Basic", "The Visual Basic Environment", "Your First Visual Basic Project", "Project Design, Forms, Command Buttons", y "Labels, Text Boxes, Variables"

HTML TUTOR PRO

Introducción estructurada a la programación con HTML. Guía completa de programación muy adecuada para jóvenes o usuarios no familiarizados con el desarrollo. El tutorial está dividido en diez secciones: fundamentos básicos, etiquetas, párrafos, imágenes, mapas, tablas, frames, formularios, sonido y vídeo.

FUENTES DE LOS ARTÍCULOS

Captura de vídeo con Delphi (I)
Los fuentes de la práctica propuesta en el artículo de portada ./program/ VIDEO1/

XML (II)
Ejemplos de algunas realizadas con elementos XML ./program/XML2/

Multimedia con Java (II)
Los fuentes de la práctica propuesta en el artículo ./program/MM2/

Aplicaciones Internet Information Server con Visual Basic (II)
Información adicional al artículo y el código de la práctica ./program/VB2/

IMPRESCINDIBLES

ANTIVIRUS

McAfee VirusScan
Panda Antivirus Platinum

GRÁFICOS

The Best Icons
LView Pro Image Processor 2.1
Paint Shop Pro 5.01 castellano
ThumbsPlus 3.30s
Xara 3D 3

INTERNET

AutoWinNet 5.5 Beta 2
CuentaPasos 3.6
CuteFTP 2.6
Eudora Light 3.0.6
Go!Zilla 3.3
HomeSite 4.0
mIRC 5.5
URL Organizer 2.0
WebZIP 2.50

MULTIMEDIA

GoldWave 4.02
WinAmp 2.09

NAVEGADORES

Netscape 4.51
Opera 3.51

RUNTIMES

Runtimes de Visual Basic

UTILIDADES

Adobe Acrobat Reader 3.01
Babylon Translator 2.20
CDR Win
Day Time Organizer 2000
DirectX 6.1
Fix 2000
Windows Commander 3.53
Win32s
WinZip 7.0

Multimedia con Java: sonido y vídeo (II)

Javier Sanz Alamillo (jsanza@teleline.es)

En este artículo se muestra cómo desarrollar aplicaciones para reproducir audio usando *Java Sound* y se detallan los fundamentos de *Java Media Player*, la otra gran baza multimedia de *Java* para reproducir audio y vídeo.

Una vez que se han mostrado las posibilidades de desarrollo con la *API Java Sound*, vamos a construir unos ejemplos con los que pondremos en práctica todo el contenido teórico mostrado. Para ello, se van a crear dos aplicaciones que reproducirán audio muestreado y ficheros *MIDI*, con lo que el programador conseguirá familiarizarse perfectamente con la *API* e incluso aventurarse a utilizar otras posibilidades como el tratamiento de *streams* o la captura de audio.

UN REPRODUCTOR DE AUDIO MUESTREADO

En este primer ejemplo se enseña cómo construir un reproductor de audio muestreado usando las posibilidades que ofrece *Java Sound*, y así reproducir por ejemplo un fichero en formato *WAV*. Una vez que los fundamentos de la *API* han sido mostrados, el desarrollador comprobará lo práctico y sencillo que es construir aplicaciones que reproduzcan audio. Posteriormente se integrarán algunas posibilidades como el control del volumen (*gain*) y el manejo de la salida en función de los altavoces (*pan*).

La reproducción de audio muestreado mediante Java Sound es una tarea simple y práctica

De todas formas, el programa se adapta para reproducir un *WAV* o *AU* indistintamente, ya que se podría reproducir cualquier formato de audio que sea muestreado sin necesidad de retoques, porque como se comentó en el artículo anterior, *Java Sound* determina lo que es necesario en función del formato que reproducir.

DESARROLLO DE LA APLICACIÓN

Para reproducir un formato de audio muestreado se realizan los siguientes pasos:

- Crear un canal para la reproducción.
- Registrar los eventos necesarios, en nuestro caso el de final de reproducción.

Para disponer de un canal de reproducción tenemos que crear un objeto del tipo *PullAudioOutput* y conectarle con un *AudioOutControlDevice*, que es un *controlDevice* adecuado, por lo que hacemos lo siguiente:

```
public class repro implements
    TransportListener {
  String fichero ;
  PullAudioOutput pullAudio-
    Output = null;
  try {
      AudioOutControlDevice
    cdevice =
```

```
        (AudioOutControlDevice)
    audioManager.getControlDevice(

    Class.forName("javax.media.so
    und.sampled.AudioOutControl-
    Device"), null);
        pullAudioOutput = cdevi-
    ce.newPullAudioOutput();
        }
     catch (UnsupportedResource-
    Exception e) {
        e.printStackTrace();
        }
     catch (ClassNotFoundExcep-
    tion e2) {
        e.printStackTrace();
        }
        ...
}
```

Se utiliza *audioManager* puesto que como se comentó, es el punto de unión con *Java Sound* y permite el acceso a los recursos necesarios para reproducir el audio. Se comprueba que exista *Java Sound* y que la salida muestreada esté soportada. Así se obtiene el objeto correspondiente *AudioOutControlDevice* que permitirá crear nuestro canal para la reproducción, con la sentencia:

```
pullAudioOutput = cdevice.
    newPullAudioOutput();
```

Una vez llegado a este punto tenemos los medios disponibles para reproducir audio muestreado. Ahora pedimos los recursos necesarios y arrancamos el canal, por tanto, ya está todo dispuesto para reproducir. Además, como se observa, la clase implementa *TransportListener* por lo que indicamos que vamos a controlar algún evento relacionado con el tratamiento del audio, en nuestro caso gestionaremos el final de reproducción. Por este motivo registramos el evento, por lo que tendremos que implementar el método *transportUpdate*() como se detallará más adelante.

```
// adquiere y arranca el canal
try {
        pullAudioOutput.acquire();
        pullAudioOutput.start();
    }
catch (ResourceUnavailableExcep-
    tion e) {
        e.printStackTrace();
}

// controlamos el fin de
    reproducción
pullAudioOutput.addTransport-
    Listener(this);
```

Seguidamente tenemos que crear un *MediaStream* mediante *AudioManager* que se encargará de iniciar los mecanismos de identificación del tipo de audio que se va a reproducir, así como información como la longitud, etc., para crear un objeto *PullMediaSource* el cual será utilizado por el canal creado. Finalmente, indicamos el formato del *stream* que leer, donde se identificará realmente el tipo de audio y el origen o *stream* de lectura de datos, que es el propio objeto *PullMediaSource,* todo ello simplemente llamando a los métodos *getFormat()* y *setSource()*:

```
try {
        InputStream is = new
    FileInputStream(fichero);
        PullMediaSource pullSource
    =
 new PullMediaSource(AudioMana-
    ger.newMediaStream(is));
        PullAudioOutput.setFor-
    mat(pullSource.getFormat());
    PullAudioOutput.setSource
    (pullSource);
    }
catch (Exception e) {
        e.printStackTrace();
}
```

Existe un pequeño truco. Seguramente le habrá llamado la atención al lector que incluso antes de definir el formato y el origen de los datos ya se ha indicado al canal que solicite los recursos y comience. Esto es debido a que se comprueba que el origen de datos esté disponible antes de iniciar la reproducción, es decir, *Java Sound* no conecta nuestro canal *PullAudioOuput* al *PullMediaSource* hasta que está seguro que los datos están disponibles y listos. Es decir, el formato es válido y se puede reproducir.

Es entonces cuando el canal recibe los recursos necesarios y se comienza su reproducción. Mediante este modelo de reproducción, las aplicaciones que utilizan el audio no necesitan preocuparse sobre la forma en que es tratado el audio y cómo se reproduce en el sistema. *Java Sound* se encarga de todo ello.

Un canal de reproducción se obtiene mediante un objeto PullAudioOutput

Ahora nos queda ver cómo reacciona a los eventos la aplicación. Tal y como se indicó al inicio del programa, la clase implementa la interfaz *TransportListener*, gracias al cual podemos detectar el evento que buscamos, como por ejemplo puede ser el de la finalización de audio.

```
public void
    transportUpdate(TransportE-
    vent e) {
  if (e instanceof EndOfMediaE-
    vent) {
    ...  // Tarea requerida
  }
}
```

El usuario puede finalizar correctamente la reproducción sin más que añadir el siguiente código:

```
addWindowListener ( new WindowA-
    dapter() {
   public void windowClosing(Win-
    dowEvent e) {
    pullAudioOutput.release();
    System.exit(0);
   }
}
```

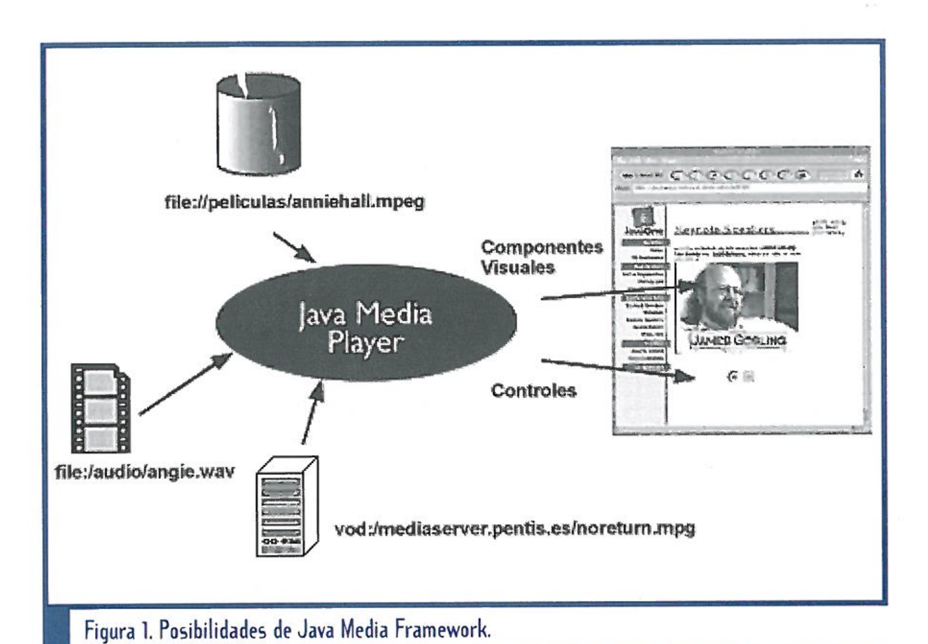


Figura 1. Posibilidades de Java Media Framework.

Recuerde que es imprescindible que el programa libere los recursos del canal, mediante la sentencia:

```
pullAudioOutput.release();
```

En función de un sistema u otro, finalizar la aplicación sin realizar una liberación de recursos puede provocar que ésta tarde más o menos tiempo en finalizar totalmente.

Como se ha podido comprobar, incluir sonido en las aplicaciones es sumamente fácil, con independencia total del tipo de archivo que se desee reproducir.

MEJORAS EN LA APLICACIÓN

Sabemos que *Java Sound* ofrece toda una serie de facilidades sobre el audio. Vamos a hacer uso de algunas de ellas. Por ejemplo, calcularemos el tiempo de duración de un *stream* de audio que vaya a ser reproducido. Como se indicó, el control se realiza en nanosegundos, por lo que simplemente tenemos que llamar a *getMediaStream().getDuration()* y así obtener el tiempo en segundos que dura el audio que reproducir. Mediante las siguientes líneas de código se puede obtener el tiempo en segundos.

```
milisegundos = pullSource.getMe-
    diaStream().getDuration().
getNanoseconds() / 1000000;
segundos = (double)(milisegundos
    / 1000.0);
```

Si quisiéramos determinar cuánto tiempo se ha reproducido únicamente se haría la llamada a *getMediaTime()*, con lo que tendríamos algo similar a lo siguiente:

```
milisegundos =
    pullAudioOutput.getMediaTi-
    me().getNanoseconds() /
    1000000;
segundos = (double)( milisegundos
    / 1000.0);
```

También podemos realizar un control sobre el volumen de reproducción o decidir qué altavoz puede sonar en un momento determinado. Mediante el uso de ***setDb*** y ***setPan*** podemos realizar estas posibilidades.

AudioManager es la unión de las aplicaciones con Java Sound

El método *setDb(float)* permite incrementar el volumen de reproducción mediante la determinación de un rango de valores *float* que va desde **-5** hasta **5**. El primer valor determina silencio, mientras que el otro se refiere al volumen máximo. El término medio **0** deja el volumen según el actual. Por ejemplo, mediante el siguiente método se reduce el volumen al mínimo:

```
pullAudioOutput.setDB((float) -5 );
```

Mediante una llamada a *setPan(float)* se puede determinar en qué proporción se reparten los altavoces la reproducción. Así por ejemplo, el valor **-10** determina que únicamente reproduce el altavoz de un lateral, el valor **0** que reproducen todos los altavoces, y **+10** reproduce al altavoz del lado contrario al anterior. Los valores intermedios reparten proporcionalmente la salida. Por ejemplo, la siguiente sentencia no altera la salida de los altavoces.

```
pullAudioOutput.setPan ((float) 0);
```

Como se ha mostrado, incluir las posibilidades para crear efectos de sonido es una tarea bastante simple.

REPRODUCCIÓN DE UN FICHERO MIDI

En este segundo ejemplo se va a construir un reproducir de ficheros de tipo *MIDI*. Mediante este ejemplo se podrán reproducir ficheros del tipo 0, 1 y *RMF*. Antes de continuar con los detalles de la aplicación, se debe mencionar que se han encontrado varios problemas en la *API* respecto al tratamiento de estos ficheros de audio, por lo que hasta que la *API* sea oficialmente lanzada, y por lo que cabe esperar, corregidos los errores, algunos programadores encontrarán que la reproducción de ficheros *MIDI* no cumple las expectativas. Por este

motivo deberán utilizar las posibilidades que ofrece *Java Media Player* para obtener una verdadera garantía.

Los problemas que suelen aparecer van desde la imposibilidad de detectar el final de la reproducción o bloqueos si no se libera el canal generado. De todos modos, una vez que la *API* sea corregida, se podrá usar sin que por ello tengamos que modificar nuestro programa.

DESARROLLO DE LA APLICACIÓN

El proceso de reproducir un *MIDI* es muy similar al anteriormente descrito. Necesitamos un canal de reproducción y registrar los eventos necesarios. En nuestro caso, al igual que el anterior, comprobaremos la finalización de la reproducción. Para disponer de un canal de reproducción tenemos que crear un objeto del tipo *SequencePlayer* y conectarle con un objeto *MidiOutControlDevice,* que es un *controlDevice* adecuado, por lo que tendremos que hacer lo siguiente:

```
SequencePlayer canal = null;
MediaStream mediaStream;
InputStream is = null;
public MidiFileApp(String fich)
     throws MediaException {
   try {
     is = new
     FileInputStream(fich);
     mediaStream =
     AudioManager.newMediaStream(is)
     ;

     // Se confirma que el fichero
     es MIDI
if ( !(mediaStream.getFormat()
     instanceof MidiFileFormat) )
     {
      throw new MediaException("No
     es un Midi");
     }
            MidiOutControlDevice
     cDevice = (MidiOutControl-
     Device)
               AudioManager.getCon-
     trolDevice(Class.forName
   javax.media.sound.midi.MidiOut-
       ControlDevice"), null);
       canal = cDevice.newSequence-
       Player();
       }
       catch (Exception e) {
        e.printStackTrace();
       }
       ...
}
```

A continuación se carga el fichero *MIDI* y se inicializa el canal, por lo que llamando al método *start()* se inicia la reproducción del *MIDI*. El código sería el siguiente:

```
try {
  canal.load(mediaStream);
  canal.acquire();
  canal.start();
 }
catch ( Exception e) {
  e.printStackTrace();
}
```

Ya sólo queda liberar los recursos cuando se ha finalizado la reproducción, invocando a *release()* de la siguiente forma:

```
canal.release();
```

La gestión de los eventos se realiza de la misma forma que en el audio muestreado, por lo que tendremos:

```
public void
    transportUpdate(Transport-
    Event e) {
  if (e instanceof EndOfMedia-
    Event) {
    salir = true ;
  }
}
```

Mediante el uso de una variable lógica detectamos el final de la reproducción y así invocamos a *release()*. Como se puede comprobar, el proceso es similar a reproducir audio muestreado, únicamente se deben usar los objetos adecuados.

Como ya se ha comentado, debido a que *Java Sound* no es una *API* totalmente liberada por *Sun*, durante la ejecución de los ejemplos podrá observar que aparecen mensajes relativos a ciertos temas. No se preocupe, su programa funcionará correctamente, y éstos no suelen ser importantes y desaparecerán cuando se libere oficialmente *Java Sound*.

En el *CD-ROM* incluido con la revista puede encontrar el código fuente completo de estos ejemplos. En el código que reproduce el fichero *WAV* se muestra el tiempo total y el transcurrido, además se producen alteraciones de volumen y cambio de reproducción en los altavoces durante unos determinados periodos de tiempo.

Es imprescindible que las aplicaciones liberen los recursos del canal mediante la llamada a release()

Si hasta ahora se ha podido comprobar el conjunto de facilidades que ofrece la *API Java Sound*, a continuación se va a describir *Java Media Framework*, y más específicamente, *Java Media Player*, que ofrece un conjunto de nuevas posibilidades, tanto en el audio, como en las ya mostradas, como en lo que parece mucho más novedoso, e interesante, poder reproducir vídeo.

JAVA MEDIA FRAMEWORK

Java Media Framework(JMF) es un conjunto de tres *APIs* desarrolladas por el grupo *JMF Working Group*, formado por *Sun, Silicon Graphics e Intel*. Este conjunto de *APIs* está constituido por *Java Media Player, Capture* y *Conferencing*.

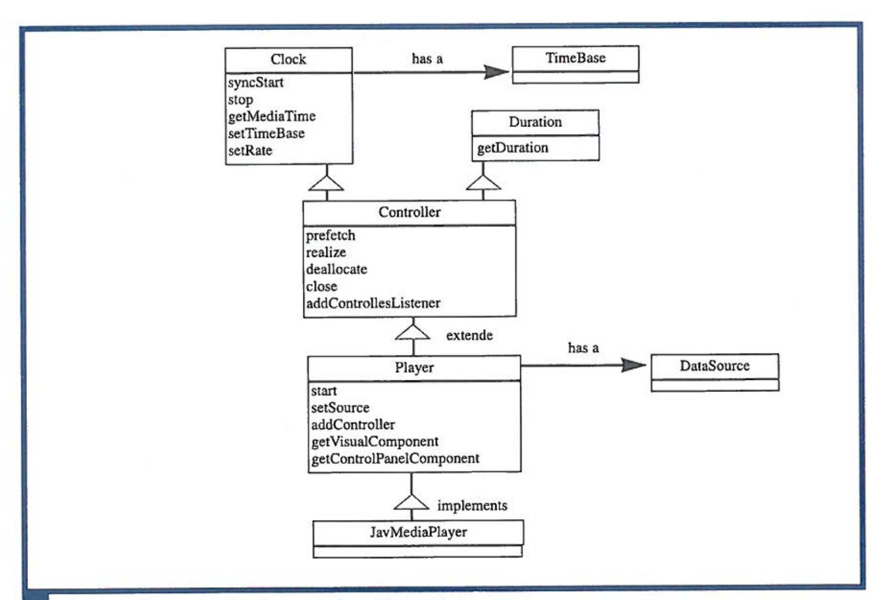


Figura 2. Diagrama de clases de Java Media Player.

Mediante el uso de *JMF* se puede visualizar, reproducir, capturar y manejar *stream* de audio y vídeo tanto en local como a través de *Internet* para realizar aplicaciones multimedia con *Java* (que pueden ser tanto aplicaciones como *applets*).

JMF se encarga de todas las tareas relacionadas con el tipo de datos que se deben manejar y permite la integración de todo un conjunto de funcionalidades a bajo nivel. Por ejemplo, implementa protocolos de clientes *pull* como *HTTP* y de servidor *push* como *RTP*.

Usando *JMF* se pueden utilizar todo un conjunto de componentes multimedia tanto en aplicaciones como en *applets*, manteniendo la compatibilidad y flexibilidad para realizar aplicaciones.

Mediante getMediaTime() se puede obtener el tiempo actual de reproducción

Las características principales de *JMF* que se pueden aplicar al conjunto de las diferentes *APIs* que lo conforman desde el punto de vista de la funcionalidad que ofrecen son las siguientes:

- Permite incorporar tecnología *Java* tanto en *applets* como en aplicaciones.
- Posibilita el uso de soluciones basadas en el *Web* mediante el uso de un entorno *Java Compatible*.
- Soporte de un amplio conjunto de formatos de ficheros, tanto de audio como de vídeo e incluso de protocolos de *streams*.

Desde el punto de vista del desarrollador, *JMF* ofrece las siguientes características.

- Los programadores de partes clientes pueden crear controles de *Java Media Player* para permitir un amplio conjunto de posibilidades mediante el uso de sencillas llamadas a algunas de las funciones.
- El uso de *JMF* permite acceder a diferentes tipos de formatos de audio y vídeo, así como la creación de algunas operaciones que se puedan aplicar a esos formatos mediante la creación de nuevos tipos de datos multimedia o reproductores, e incluso entradas de datos multimedia.

JAVA MEDIA PLAYER

Si *JMF* ofrece un gran conjunto de posibilidades para el tratamiento multimedia, *Java Media Player* permite trabajar con la mayoría de los estándares de audio y vídeo, como son *MPEG-1, MPEG-2, QUICKTIME, AVI, WAV, AU* y *MIDI*. Usando *Java Media Player*, se pueden sincronizar y gestionar los datos multimedia de distintas fuentes (red, local, etc.).

Ya que los actuales reproductores de vídeo y de audio son dependientes del sistema en el que se ejecutan, la *API* de *JMF* ofrece un nivel de abstracción que oculta los detalles de la implementación al programador. Así, un reproductor construido mediante el uso de esta *API* ofrecerá la independencia de plataforma tan deseada al utilizar *Java*. Aunque internamente *JMF* utilizará métodos nativos para acceder a esas posibilidades multimedia, las aplicaciones o *applets* que los utilicen no tienen que conocer como están implementados. Las características principales de *JMF Player API* se pueden resumir en:

- Ofrece una serie de mecanismos de gestión y control de diferentes protocolos.
- Gestión y control de diferentes tipos de datos multimedia, tanto de audio como de vídeo.
- Un modelo de gestión de eventos asíncronos entre la *API* de *Java Media Player* y las aplicaciones y los *applets*.

FUNDAMENTOS DE JAVA MEDIA PLAYER

A la hora de utilizar la *API Java Media Player* es necesario tener unos conocimientos básicos sobre los obje-

tos y estructuras de datos que utilizar, los eventos que se producen y el conjunto de las llamadas a la *API*, así como su funcionamiento.

OBJETOS BÁSICOS DE LA API

Un objeto del tipo *DataSource* encapsula la localización del tipo de datos que utilizar así como el protocolo y el *software* necesario para utilizarlo. Un objeto del tipo *Player* contiene un *DataSource*. Una vez que el objeto *DataSource* es utilizado, no se puede aplicar sobre él otro tipo de datos multimedia.

Un objeto *Player* es identificado mediante un objeto *MediaLocator* o un objeto *URL* (*Universal Resource Locator*). *MediaLocator* es una clase de *JMF* que describe el tipo de datos multimedia que un objeto *Player* va a reproducir. Un objeto *MediaLocator* es parecido a *URL* y puede ser construido mediante un *URL*.

Con Java Media Framework (JMF) se puede reproducir audio, vídeo, realizar capturas, etc.

Un objeto *Player* puede tratar diferentes tipos de datos multimedia con independencia del tipo de fuente desde donde se reciban, como algo local, mediante red, etc. *JMF* soporta dos tipos de fuentes de datos:

- El definido como *Pull Data-Source*. El cliente inicia los datos que transferir y el control del flujo de datos desde el origen de datos. Protocolos de este tipo son por ejemplo *HTTP* (*Hypertext Transfer Protocol*) y *FILE*.
- En *Push Data-Source*, el servidor inicia la transferencia de datos y controla el flujo de los mismos. Ejemplo de este tipo de fuentes son el vídeo por demanda (*VOD*), *multicast media*, *broadcast media*. Dentro del *broadcast media* esté el protocolo *RPT* (*Real-Time Transport Protocol*), desarrollado por el *IETF* (*Internet Engineering Task Force*). El protocolo *MediaBase* desarrollado por *SGI* es utilizado para *VOD*.

El grado de control de la parte cliente está en función de las necesidades del usuario y dependerá también de las posibilidades del tipo de datos que gestionar. Por ejemplo, un fichero *MPEG* permite que un usuario pueda repetir la reproducción del mismo o localizar una secuencia determinada dentro del fichero de vídeo.

Por otro lado, *broadcast media* define que el servidor realiza el control de los datos por lo que el usuario no tiene ningún control sobre la reproducción. En algunos protocolos *VOD* el usuario tiene acceso a algunos controles básicos, por ejemplo, se le permite localizar cierta secuencia, pero no pueden retroceder ni adelantar secuencias.

GESTIÓN DE LA REPRODUCCIÓN MEDIANTE PLAYER

Los objetos de tipo *Player* se encargan del tratamiento del *stream* de datos en función del tiempo, leyendo los datos de un objeto *DataSource* y manipulándolo en el momento adecuado. Un objeto *Player* implementa una interfaz *Player*. En la figura 2 se muestra un diagrama de clases en el que se muestran las relaciones entre los principales objetos utilizados para la reproducción mediante *Java Media Player*.

Un objeto del tipo *Clock* define las operaciones básicas de sincronización y control de tiempo de reproducción que un objeto *Player* utilizará para el control de la reproducción de datos multimedia. Los objetos del tipo *Controller* son derivados de *Clock* y ofrecen métodos para la gestión de los recursos del sistema y la precarga de los datos, así como los mecanismos necesarios para tratar los eventos en la reproducción.

Un objeto del tipo *Duration* ofrece una forma de determinar la duración de

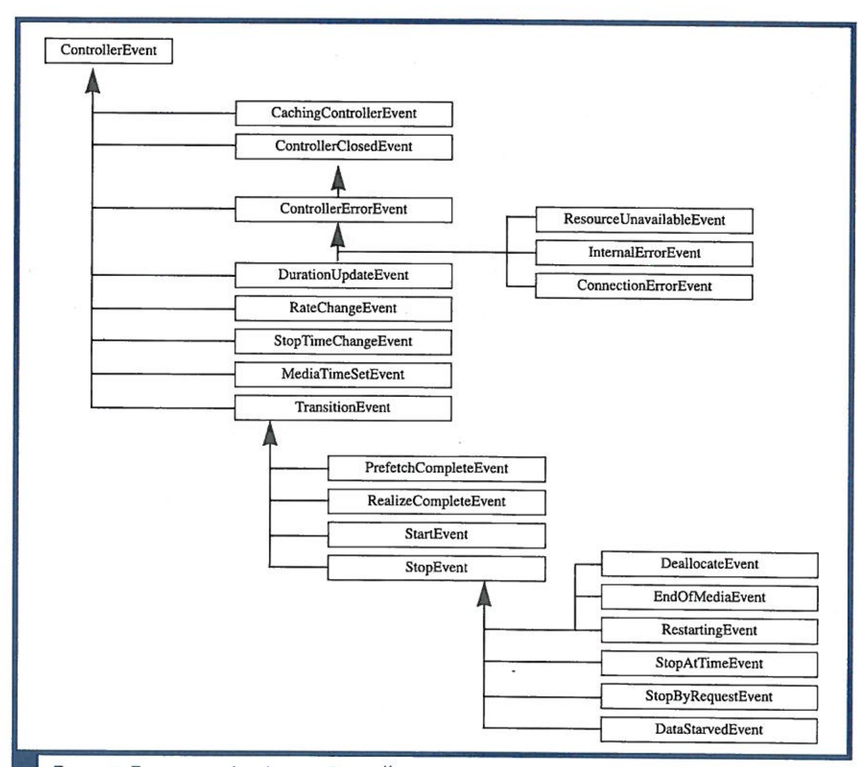


Figura 3. Eventos producidos por Controller.

la reproducción que se está ejecutando. Además, los objetos *Player* ofrecen un conjunto de controles estándar sobre la reproducción.

Cada objeto tipo *Player* consta de un objeto *TimeBase* que define el control del tiempo para un objeto *Player*. Cuando un objeto *Player* es iniciado, *TimeBase* se encarga de todos los procesos necesarios de sincronización.

Un interfaz de usuario para un objeto tipo *Player* incluye una parte visual, un componente y una parte de control, mediante unos controles del estilo panel visual. El desarrollador puede elegir entre personalizar la interfaz del usuario para un objeto tipo *Player* o utilizar la interfaz por defecto.

Java Media Player permite reproducir la mayoría de los formatos actuales de audio y vídeo

Un objeto del tipo *Player* debe realizar una serie de tareas antes de poder reproducir el formato deseado. Debido a que esas operaciones consumen un determinado tiempo, *JMF* permite el control de saber qué ocurre en este periodo de tiempo mediante la definición de unos estados de operación para cada objeto *Player*, permitiendo así un mecanismo de control para cada cambio de estado.

Estos estados serán detallados con más profundidad en el siguiente artículo, ya que son necesarios para comprender la construcción de un reproductor.

EVENTOS

La gestión de los eventos de *JMF* permite a los programas controlar condiciones de error, como recursos no disponibles, *out-of-data*, etc.

Los eventos también ofrecen un protocolo básico de notificación: cuando un programa realiza una llamada a un método asíncrono de un objeto *Player*, el programa sólo puede estar seguro de que una operación ha sido completada cuando recibe el evento adecuado. Se definen dos tipos de eventos: *GainControl* y *Controller*.

Un objeto *GainControl* ofrece la gestión de un único evento denominado *GainChangeEvent*. Para responder a los cambios en el control de volumen, se debe implementar la interfaz *GainChangeListener*.

Un *Controller* puede controlar una variedad de eventos derivados de *ControllerEvent*. Para tratar los eventos de un objeto *Controller* se tiene que implementar la interfaz *ControllerListener*. En la figura 3 se muestran los eventos que pueden ser tratados mediante *Controller*.

Los eventos del tipo *ControllerEvents* se dividen a su vez en tres categorías que son notificaciones de cambio, eventos de finalización y eventos de transición.

Los eventos de notificación de cambio son *RateChangeEvent* y *DurationUpdateEvent* e indican que algunos atributos de *Player* han sufrido alguna modificación, a menudo como respuesta de alguna ejecución de un método. Por ejemplo, un objeto *Player* lanza un *RateChangeEvent* cuando el atributo *rate* es cambiado mediante el uso de *setRate()*.

Los eventos del tipo *TransitionEvents* o eventos de transición permiten que un programa controle los cambios en el estado de un objeto *Player*. Los diferentes estados deben ser controlados para evitar ciertos problemas como condiciones de carrera en algunos métodos de *Player*.

Los eventos de finalización, *ControllerClosedEvents*, son gestionados por un objeto *Player* cuando éste finaliza. Un evento especial, *ControllerErrorEvent* es un caso específico de evento tipo *ControllerClosedEvents* y se encarga del control de errores en la reproducción, intentado ofrecer un medio de control de errores a las aplicaciones.

Como se puede observar, el conjunto de eventos permite un control total en la reproducción de datos multimedia, tanto en audio, vídeo, capturas, etc.

En el próximo artículo se mostrarán los diferentes estados de un objeto *Player* con más detalle, las llamadas a la *API* de *JMF* y mediante la creación de un reproductor de vídeo *MPEG* se detallará más específicamente la creación de un reproductor multimedia general, sus posibilidades y el uso de los diferentes controles que se pueden aplicar.

CONCLUSIÓN

Como se ha podido comprobar a lo largo del artículo, *Java Sound* ofrece un amplio conjunto de facilidades para construir aplicaciones que utilicen audio. Mediante los ejemplos se ha demostrado y cómo reproducir audio tanto muestreado como *MIDI*, comprobándose que no se trata de una tarea demasiado difícil ni compleja de llevar a cabo.

En el siguiente artículo se completará la descripción de los conceptos básicos de *Java Media Player* y se construirá un reproductor de vídeo usando las todas las posibilidades disponibles, además de detallarse más conceptos de la *API* que son importantes para añadir nuevas posibilidades y comprender el funcionamiento del reproductor.

Aplicaciones IIS (y II)

Jordi Agost (agost@eup.udl.es)

En este artículo aprovecharemos nuestros conocimientos para programar los servidores de *Internet* y acercar las posibilidades de las aplicaciones *IIS*.

Veamos en primer lugar una de las principales formas de controlar la navegación del usuario por la aplicación. Este factor es muy importante en las aplicaciones de *Internet* ya que no estamos nunca seguros de la opción que elegirá el usuario.

DESPLAZAMIENTO A TRAVÉS DE LA APLICACIÓN

Uno de los grandes retos de las aplicaciones *IIS* es poder controlar la forma en que el usuario se desplazará por la aplicación ya que es casi imposible predecir cuándo dicho usuario irá a un hipervínculo del historial o hará *clic* en el botón de atrás. Para evitar un poco el "caos circulatorio" se ha incluido la propiedad *URLData* en las aplicaciones *IIS*. Mediante esta propiedad podemos evitar que el usuario pase de la página número uno a la página número tres sin haber pasado antes por la dos (siempre y cuando nos interese realizar dicha acción). Imaginemos que tenemos un objeto de tipo *WebClass* compuesto por tres diferentes elementos de *Web*. La propiedad *URLData* permitirá establecer un número de una secuencia determinada para cada petición de la clase de *Web*, que se haga por parte del explorador.

La propiedad URLData evita que el usuario vaya de la página uno a la página tres sin pasar por la dos

Lo que en realidad hará la propiedad *URLData* es establecer o devolver (dependiendo de su uso en el código) un parámetro que se agregará al final de cada dirección *URL* generada. En el momento en que establecemos la propiedad, el archivo *DLL* en tiempo de ejecución analizará el objeto de datos y agregará el valor que hemos asignado a la propiedad *URLData* a cada dirección *URL* que encuentre incrustada en una respuesta. Por ejemplo imaginemos:

```
WCI=webitem1?WCE=event1
```

Y establecemos el valor de la propiedad URLData:

```
Me.URLData = 01
```

El archivo *DLL* en tiempo de ejecución agregará el valor de la propiedad *URLData* a la dirección *URL*, utilizando el indicador *WCU*. El código *HTML* resultado de esta operación sería el siguiente:

```
WCI=webitem1?WCE=event1&WCU=01
```

En la siguiente petición el explorador incluirá el argumento, indicando así al servidor que se trata de la siguiente petición de secuencia después de la respuesta **01**. En este punto, y debido a que el usuario ya ha realizado una segunda respuesta deberíamos aumentar la propiedad *URLData* en uno, ya que estamos en la segunda petición. Todo esto lo podríamos realizar sin problemas con el siguiente código:

```
Me.URLData = 02
```

De nuevo cuando la clase de *Web* encontrase este valor en la propiedad *URLData* agregaría este número a todas las direcciones *URL* y dicho número lo enviaríamos y almacenaríamos en el explorador, y del mismo modo que antes, sería devuelto en la siguiente petición. Una vez solucionado el problema de la navegación veamos ahora los componentes necesarios en la generación de aplicaciones *IIS*.

GENERACIÓN DE APLICACIONES IIS

Cuando generemos una aplicación *IIS* tendremos que crearla como un archivo *DLL* (o un componente en proceso). Esto se realiza a través la opción **Generar** del menú **Archivo.** Cuando llevemos a cabo dicha opción, *Visual Basic* generará los siguientes archivos:

- Un archivo *.DLL*: es el archivo empleado por *Visual Basic* para ejecutar el proyecto e incluye el objeto *WebClass* y la biblioteca en tiempo de ejecución de *Visual Basic*.
- Un archivo *.ASP*: será el responsable de albergar la aplicación en el explorador y de crear el componente en tiempo de ejecución.
- Un archivo *.EXP*: generado por el enlazador.
- Un archivo *.VBW*: con información sobre el diseño de ventanas para el proyecto.

PASOS PARA REALIZAR UN PROYECTO

Recordemos los pasos básicos que deberemos realizar para desarrollar una aplicación de tipo *IIS* dentro del entorno de *Visual Basic*:

1. Iniciar un nuevo proyecto y seleccionar **Aplicación IIS** tal y como se aprecia en la figura 1.
2. Guardar el proyecto, ya que de otro modo no sería posible añadir ciertos elementos posteriormente.
3. Añadir a la *clase Web* todos los elementos *Web* necesarios, bien sean plantillas *HTML* o elementos personalizados de *Web*.
4. Agregar más tarde los eventos personalizados y escribir código para los eventos (ya sean estos personalizados, estándar o del proyecto).
5. Añadir a continuación, si son necesarios, otros módulos de código u objetos de tipo *WebClass*.
6. Depurar la aplicación y visualizarla desde varios exploradores.
7. Finalmente compilaremos la aplicación y la distribuiremos en la Red.



Figura 1. Selección del tipo de aplicación IIS.

Vamos ahora a seguir estos pasos para así ver una aplicación de ejemplo.

INICIANDO UN PROYECTO

En primer lugar, tal y como hemos dicho crearemos un nuevo proyecto de *Visual Basic* y cuando llegue el momento de introducir el tipo de proyecto seleccionaremos **Aplicación IIS**. A continuación grabaremos el diseñador bajo el nombre de **IIS1** (el sistema le añadirá la extensión *.DSR*). Después tendremos que grabar el proyecto al que se le asignará la extensión *.VBP* con el nombre **IISproj.vbp**. Acto seguido se activa el evento *Start* del objeto *clase Web* automáticamente (recordemos que si desde el explorador no se especifica un objeto *WebItem* o un evento concreto en la solicitud, automáticamente se activa el evento *Start*.

Deberemos guardar el proyecto antes de incluir una plantilla

De este modo el diseñador de *clase Web* ha añadido el siguiente código:

```
Private Sub WebClass_Start()
    'Escribir una respuesta al
     usuario
    With Response
        .Write "<HTML>"
        .Write "<body>"
        .Write "<h1><font
     face=""Arial"">Página inicial
     de WebClass1</font></h1>"
        .Write "<p>Esta respuesta
     se ha creado en el evento
     Start de WebClass1.</p>"
        .Write "</body>"
        .Write "</HTML>"
    End With
End Sub
```

Con lo que si ahora ejecutamos el proyecto veremos:

```
Página inicial de WebClass1
Esta respuesta se ha creado en el
     evento Start de WebClass1
```

LAS PLANTILLAS HTML EN LA CLASE WEB

Las plantillas *HTML* que agregamos a una determinada aplicación de tipo *IIS* permitirán a la *clase Web* enviar las páginas *HTML* al explorador, respondiendo así a la petición de un usuario. En el momento de agregar una plantilla (deberemos agregar una por cada página *HTML* que deseemos mostrar) deberemos elegir a su vez una página *HTML* asociada a ella.

Entonces *Visual Basic* examinará dicha página para buscar etiquetas de formulario, de imagen, hipervínculos, etc., es decir, cualquier tipo de etiqueta que contenga una dirección *URL*. Los atributos de las etiquetas de las páginas *HTML* que pueden iniciar una petición del servidor se muestran a la derecha del panel derecho del diseñador de clases *Web*. Deberemos guardar el proyecto antes de incluir una plantilla. Posteriormente, cuando *Visual Basic* añada una plantilla, éste comprobará si el archivo elegido se encuentra fuera del directorio del proyecto, en cuyo caso *Visual Basic* realizará una copia del archivo en el directorio del proyecto.

Suponiendo que en el directorio exista un archivo con el mismo nombre o bien, dicho archivo ya existiera en el directorio, *Visual Basic* añadirá un número al nombre del archivo. A partir de este punto la copia del archivo *HTML* es el origen del proyecto. Si queremos efectuar cambios en la apariencia de la página *HTML* lo deberemos hacer sobre este archivo y no sobre su localización original. Para verlo generamos la siguiente plantilla *HTML* a la que llamaremos **PRUEBA.HTM:**

```
<HTML>
<head>
<title></title>
</head>
<body>
<h1>Prueba </h1>
<p><br>
<a HREF="http://www.nose.com">Link
    a algún sitio! </a></p>
<p><img SCR="\hola.bmp"
    src="hola.jpg" width="98"
    height="47"></p>
</body>
</HTML>
```

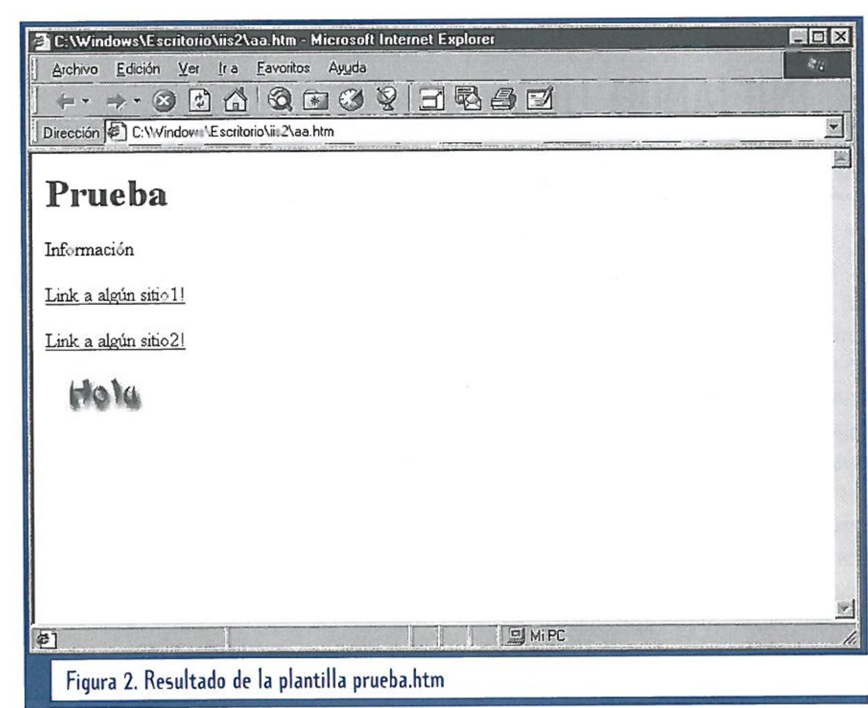


Figura 2. Resultado de la plantilla prueba.htm

y en el mismo directorio almacenamos también un gráfico cualquiera que se llame **"hola.jpg"**. Este listado *HTML* no tiene nada de especial hasta que lo importamos dentro de *Visual Basic*. Una vez que en *Visual Basic* hemos abierto un nuevo proyecto *IIS* e importamos un archivo *HTML* dicho archivo pasa ahora a ser un elemento de *Web* y se le asigna un nombre por defecto **Template1.**

Cuando importamos un archivo HTML dicho archivo pasa a ser un elemento de Web

Tal y como hemos dicho en el panel derecho de la ventana del diseñador de *clases Web* aparecerá una lista de todas las etiquetas *HTML* que son susceptibles de conectarse a un evento de *Visual Basic*. En cierto modo es algo parecido al proceso estándar de colocación de un control dentro de un formulario de *Visual Basic*. Por ejemplo si colocamos una caja de texto o control *TextBox* dentro de un formulario estaremos colocando en realidad un objeto dentro de un contenedor. Y una vez realizado este paso podremos acceder a todas las propiedades y eventos desde la ventana de código del formulario. Así pues nuestro **Template1** es un objeto de la *clase Web*, tal y como el objeto *TextBox* es un objeto de un formulario. Observemos ahora el hipervínculo que añadimos a la plantilla. Dicho hipervínculo se importó dentro del diseñador de *clase Web* con el nombre de **Hyperlink1**. De este modo pulsamos encima del botón de la derecha y elegimos **Conectar con el evento personalizado.** Aparecerá entonces en la ventana del diseñador un nuevo evento llamado **HyperLink1.** Al hacer doble clic encima de este evento veremos la ventana de código con el evento siguiente:

```
Private Sub
    Template1_Hyperlink1()
End Sub
```

Llegados a este punto, este evento se desencadenará cada vez que el usuario pulse el hipervínculo. Añadimos ahora en el evento la sentencia:

```
Private Sub
    Template1_Hyperlink1()
  Response.Write "<BR>
    Hipervínculo Activado </BR>"
End Sub
```

Ejecutamos el proyecto y cambiamos el nombre del archivo que se encuentra al final de la dirección *URL*.

Al generar una aplicación IIS debemos crearla como un archivo DLL

Si examinamos el código veremos como el hipervínculo ya no apunta a la dirección que habíamos utilizado en un principio (**www.nose.com**) sino que ahora señala a un evento de la *clase Web*. Al pulsar el hipervínculo veremos las palabras **"Hipervínculo Activado"** en el navegador, que es el evento que tenemos en el código de *Visual Basic*.

ESCRITURA DE PLANTILLAS

De igual forma que acabamos de enviar la línea de respuesta al navegador, también podemos enviar plantillas.

Iremos ahora al evento *Start* de la *clase Web* (hay algunas líneas que el diseñador de *clase Web* ha incluido por defecto). Borraremos todo lo que haya escrito y lo sustituiremos por la siguiente sentencia:

```
Private Sub WebClass_Start()
WebClass.Template1.WriteTemplate
End Sub
```

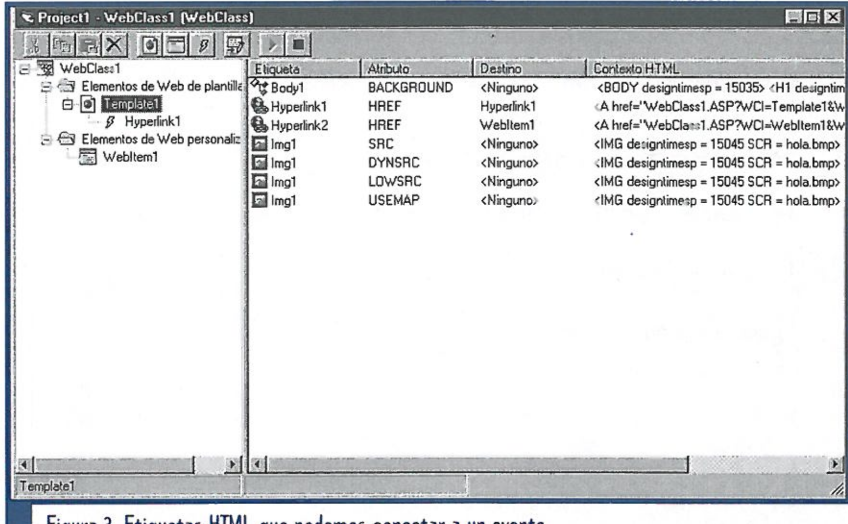


Figura 3. Etiquetas HTML que podemos conectar a un evento.

Ahora ejecutamos la aplicación al igual que en el paso anterior pero con la diferencia de que lo primero que vamos a ver va a ser la plantilla que hemos escrito anteriormente.

EL EVENTO WRITETEMPLATE UNIDO AL PROCESSTAG

Una de las ventajas que tienen las aplicaciones *IIS* dentro de *Visual Basic* es la posibilidad de unir el evento *WriteTemplate* con el evento *ProcessTag*. Con esta unión se pretende crear un efecto similar al que se produce, al usar la instrucción *ResponseWrite* de una página *ASP* para generar *HTML* de una forma dinámica. Vamos a ver un pequeño ejemplo de este proceso. Para ello deberemos abrir el archivo de plantilla que se encuentra en el directorio del proyecto añadiremos la siguiente línea de código:

```
<WC@EtiquetaPrueba> Información
    </WC@EtiquetaPrueba>
```

Con esta línea hemos conseguido crear una etiqueta "personalizada", de nombre **Etiqueta Prueba**. Ahora, durante la ejecución del método *WriteTemplate* cada vez que *Visual Basic* se encuentre con una etiqueta con el prefijo **WC@** se lanzará el evento *ProcessTag*. Dicho evento tiene la siguiente sintaxis:

```
Private Sub
    Template1_ProcessTag(ByVal
    TagName As String,
TagContents As String, SendTags
    As Boolean)
```

Donde *TagName* es el nombre de la etiqueta (en nuestro caso sería **EtiquetaPrueba**) y *TagContents* es el contenido de la misma (en nuestro caso **Información**). Estos argumentos pueden ser modificados como queramos para así devolver al navegador el valor que nos interese. Para verlo añadamos el siguiente código al evento *ProcessTag*:

```
Private Sub
    Template1_ProcessTag(ByVal
    TagName As String,
              TagContents As
    String, SendTags As Boolean)
 If TagName = "WC@EtiquetaPrueba"
    Then
   TagContents "<B> Contenido
    reemplazado! </B>"
 End If
End Sub
```

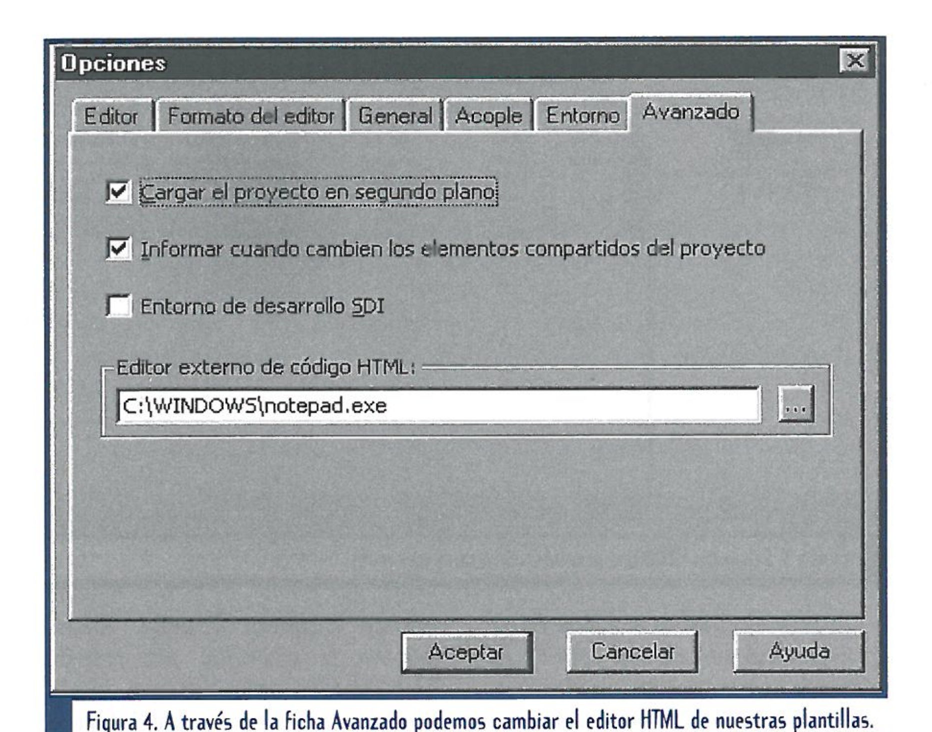


Figura 4. A través de la ficha Avanzado podemos cambiar el editor HTML de nuestras plantillas.

Si ahora ejecutamos la aplicación veremos que la información original (**Información**) ha sido reemplazada por la nueva frase que hemos añadido.

ELEMENTOS PERSONALIZADOS DE WEB

Dentro de la *clase Web* también podemos añadir *elementos* personalizados, los cuales son similares a los eventos que hemos visto para una plantilla *HTML* pero con la diferencia, tan sólo existe código *Visual Basic* asociado.

```
Los eventos de Web personalizados
     son eventos pero sin planti-
     lla, tan sólo existe código
     Visual Basic agrupado
```

Agregamos un elementos *Web* personalizado, que por defecto tendrá el nombre **WebItem1**. A continuación modificamos la plantilla y añadimos el hipervínculo. Así escribiremos:

```
A HREF="http://www.nose2.com">
     Link a algún sitio2!</A>
```

Entonces veremos que en la ventana del diseñador ha aparecido, en la sección de la plantilla *HTML*, un segundo hipervínculo con el valor **"<Ninguno>"** en la sección de destino. Pulsaremos con el botón de la derecha del ratón encima del segundo hipervínculo y veremos cómo aparece muestra un menú contextual. Escogemos la opción **Conectar con elemento de Web** y en el cuadro de diálogo que se muestra elegimos **WebItem1**.

A partir de este punto observaremos en el panel derecho del diseñador de *clases Web* (suponiendo que tengamos seleccionado las plantillas *HTML*) como el destino del segundo hipervínculo ha cambiado y ahora es **WebItem1**. Hacemos doble clic en el elemento de *Web* para ver el código asociado al mismo y en el evento *Respond* (que saldrá por defecto) escribimos el siguiente código:

```
Private Sub WebItem1_Respond()
  With Response
    .Write "<B> Contenido
     reemplazado! </B>"
    .Write "<B> Contenido
     reemplazado! </B>"
    .Write "<B> Contenido
     reemplazado! </B>"
  End With
End Sub
```

Acto seguido ejecutaremos la aplicación y observaremos que cuando hacemos clic sobre el segundo hipervínculo aparece la siguiente línea en pantalla:

```
Contenido reemplazado! Contenido
     reemplazado! Contenido
     reemplazado!
```

Lo primero que se ejecutará en el programa es el evento Start

Cuando hacemos clic en el segundo hipervínculo encontramos que se lanza el evento. Una vez conocido el proceso de creación de una aplicación *IIS*, vamos a ver las opciones que tenemos para depurarla.

DEPURAR UNA APLICACIÓN IIS

Para depurar una aplicación tipo *IIS* dentro del entorno de *Visual Basic 6* se utiliza el mismo proceso que con otra aplicación, es decir activando el modo de ejecución en *Visual Basic*.

EL PROCESO

Visual Basic creará la raíz virtual donde se ejecutará el archivo *.ASP* de la aplicación y suponiendo que sea necesario iniciaría el explorador predeterminado del sistema con una referencia a este archivo *.ASP* Simultáneamente este archivo iniciará la clase *Web* iniciando *Internet Explorer*, yendo a la dirección

URL de nuestra aplicación y activando el evento *BeginRequest* de la clase *Web*.

DIRECTORIOS VIRTUALES

Cuando iniciamos el proceso de depuración, *Visual Basic* nos informará de que va a crear un directorio virtual para nuestra aplicación. Este directorio se utiliza para publicar contenido en *Internet* (en el *Web*) a partir de directorios que se encuentran fuera del directorio de inicio del servidor. Una vez especificado dicho directorio no podrá ser cambiado.

LOS ERRORES EN LAS CLASES DE WEB

Dentro de las aplicaciones *IIS* también podemos utilizar las sentencias de control de errores que incorpora *Visual Basic*, así como el objeto *Err* y sus propiedades. Sumadas a estas características tenemos, siempre dentro de las aplicaciones *IIS*, dos características especiales para el control de errores:

El método Trace ayuda a depurar errores, analiza el rendimiento y los datos estadísticos

- El método *Trace*, que sirve para la depuración de la aplicación en el equipo en que ésta se desarrolla. Ayuda a depurar errores, analiza el rendimiento y los datos estadísticos.
- El evento *FatalErrorResponse* para reaccionar ante errores graves en tiempo de ejecución o errores fatales. Un error fatal es un error irrecuperable de la aplicación o de un cierto elemento de *Web*. Un error que no admite marcha atrás y a partir del cual nos es imposible restaurar el estado anterior de la aplicación. Si sucede un error de este tipo la aplicación lanza el evento *FatalErrorResponse*, finalizando la aplicación a continuación y destruyéndose la instancia de la *clase Web*. Si un error de este tipo sucede, desde la aplicación aún podemos dirigir un mensaje al objeto *Response* para decirle algo al usuario (lo haremos desde el evento nombrado, *FatalErrorResponse*). Dicho evento posee un argumento, *senddefault*, el cual si posee el valor **False** nos permitirá escribir nuestro propio mensaje de error en el objeto *Response*, y si posee el valor **True** aparecerá el mensaje de error predefinido del archivo *.ASP* que en ese momento esté asociado.

DISTRIBUCIÓN DE UNA APLICACIÓN IIS

Y como paso final de nuestra aplicación tan sólo queda distribuirla. El mejor modo de realizar este paso consiste en utilizar el **Asistente de empaquetado y distribución de Visual Basic**. Dicho asistente empaquetará todos los archivos asociados de nuestro proyecto en un archivo *.CAB*. Es un archivo que a su vez contiene los archivos necesarios para realizar una descarga de los diferentes componentes en *Internet*, la cual será realizada por el explorador del usuario que descomprimirá el archivo *.CAB* e instalará el componente pertinente).

Hay que tener en cuenta que aunque las aplicaciones *IIS* se instalan en un servidor *Web* deberíamos actuar de igual modo que si lo hiciéramos en un equipo cliente cualquiera. Es decir, el archivo *.cab* deberá descomprimirse en el servidor para que acto seguido se instalen y registren los componentes necesarios para el buen funcionamiento de nuestra aplicación. Y dicha descompresión o distribución del archivo *.CAB* se debe realizar obligatoriamente en un directorio virtual. Resumiendo podemos decir que los pasos que debemos seguir para que nuestra aplicación sea operativa son:

1. Compilar la aplicación.
2. Utilizar el *Asistente de empaquetado y distribución* para generar un archivo *.CAB* con todos los archivos necesarios para la ejecución.
3. Utilizar el mismo asistente para distribuir la aplicación en el servidor *Web* que deseemos.

Como comentamos el *Asistente de empaquetado y distribución* creará un archivo *.CAB* que contendrá los diferentes archivos *.DLL* del proyecto.

Recordemos que el archivo *.dsr* es en el que se almacena la información de los elementos, eventos y propiedades de los objetos de la aplicación *IIS*, y el archivo *.DSX* almacena información binaria de la aplicación.

Pero puede ser que para ejecutar nuestro proyecto, necesitemos algunos archivos asociados que no quedarán incluidos dentro del archivo *.CAB*, como por ejemplo las páginas *HTML* asociadas al proyecto. Dichas páginas no estarán empaquetadas dentro del archivo *.CAB*, sino que serán copiadas en el sitio *Web*.

Es importante que junto a estas páginas distribuyamos todos los archivos suplementarios que estas necesiten, como por ejemplo archivos de imágenes, ya que el asistente no detecta los archivos que dependen de las páginas *HTML* y no los incluirá por defecto si no se lo indicamos.

Hemos visto en estos artículos la base teórica y una aplicación práctica que nos servirá para crear nuestras aplicaciones sobre un servidor de *Internet*, modificando la respuesta del servidor según nos convenga. Así podremos orientar aplicaciones ya existentes a *Internet* o crear aplicaciones totalmente nuevas diseñadas específicamente para la *Web*.

Desarrollo cliente/servidor (I)

Javier Toledo (jtoledo@sei.es)

A medida que la empresa crece, también lo hacen los requerimientos de cada momento y por lo tanto las aplicaciones. Una buena solución al problema que puede surgir consiste en la adopción de un sistema cliente/servidor.

¿QUÉ ES CLIENTE /SERVIDOR?

Se trata de un sistema en el que se distribuye la carga de procesamiento en diversas aplicaciones. En lugar de construir una aplicación sobre la que recaiga todo el peso de atender a un grupo de usuarios, se creará una aplicación en la que se repartirá la carga. Por una parte la aplicación servidor recibirá la mayor parte del procesamiento mientras que las aplicaciones clientes se encargarán de efectuar las tareas sencillas, aliviando la carga del servidor.

Cliente/servidor es un sistema donde la carga de procesamiento se reparte en diversas aplicaciones

Un sistema cliente/servidor suele ejecutarse en sistemas distintos, en uno de ellos correrá la aplicación servidor y en los restantes las diversas aplicaciones clientes. Con ello en lugar de invertir nuestro presupuesto para *hardware* en un *PC* de grandes prestaciones para cada uno de los usuarios de nuestras aplicaciones, deberemos invertirlo en la compra de modestos ordenadores que se encargarán de ejecutar las aplicaciones clientes y un ordenador de grandes prestaciones sobre el que correrá la aplicación servidor.

Así, una buena arquitectura cliente/servidor proporciona:

- Un menor tiempo de desarrollo.
- Mayor tiempo de vida de las aplicaciones.
- Las instalaciones cliente/servidor permiten diversas configuraciones para los servidores de datos y para los clientes, sin modificar la programación de dichos clientes, ni alterar los datos del servidor.
- Aumento del rendimiento y reducción de costes gracias a la distribución de la carga de la máquina.

COMPONENTES CLIENTE/SERVIDOR

El cliente es la aplicación que interactúa a su vez con el usuario, consta de lo siguiente:

- Posee una *interfaz* gráfica.
- Realiza las validaciones de los datos ingresados.
- Hace pedidos a el/los servidores.
- Verifica la "lógica del negocio".

El servidor en general es un administrador de base de datos y/o de otros recursos que a su vez son compartidos.

- Administra la base de datos.
- Verifica la integridad de la base de datos y la seguridad.
- Ejecuta parte de la lógica de la aplicación
- Responde las peticiones de los clientes.

EL MEDIO DE UNIÓN

Provee de la interconexión que es necesaria entre los clientes y los servidores.

- Protocolos de comunicación entre los equipos (*IPX/SPX*, *TCP/IP*, etc.)
- *RPC, Sockets, named pipes*, etc.
- Servicios intermedios: *SQL*, correo electrónico.

DEFINICIÓN DE PROCESOS EN MODO CLIENTE/SERVIDOR

- Se obtiene cliente/servidor cuando el programa cliente está en otro espacio de memoria que el servidor (si el cliente corre sobre una máquina distinta a la que utilizamos para el servidor es obvio que esto se cumple).
- El programa del cliente es independiente del programa servidor, cualquiera de ellos puede ser cambiado sin que se vea afectado el otro.

En un sistema cliente/servidor real el primero realiza peticiones utilizando una red al segundo

- El servidor puede atender a múltiples clientes al mismo tiempo.
- En general consideramos que cliente/servidor involucra redes.
- Una parte de la lógica de la aplicación se ejecuta en el cliente.
- Usualmente las acciones las inicia el cliente.
- Los clientes poseen interfaces gráficas.
- Los servidores suelen poseen bases de datos y proveen protección y seguridad de datos.

UTILIZACIÓN DE BASES DE DATOS

Un desarrollo cliente/servidor típico es una aplicación consistente en un cliente trabajando contra un servidor de bases de datos del tipo *SQL Server* u *Oracle*. No todos los desarrollos contra bases de datos son desarrollos clientes/servidor. Las aplicaciones realizadas para utilizar una base de datos de "escritorio" del tipo de *Access* son aplicaciones sobre las que recae toda la tarea de procesamiento. En un sistema cliente/servidor real el cliente realiza peticiones generalmente utilizando una red al servidor, y es este último el encargado de procesar esa solicitud y devolver la información al cliente.

En los sucesivos artículos que formarán la serie presentaremos el código fuente de una aplicación c/s que trabaja contra un servidor de bases de datos. Así mismo se seguirá el proceso de diseño y creación de la base de datos, tanto en *SQL Server* como en *Oracle*. También desarrollaremos una aplicación c/s orientada a *Internet*.

FASES DEL DISEÑO

Uno de los primeros pasos que hemos de dar en el diseño de un sistema c/s es crear un buen diseño. Para ello hemos de tener muy en cuenta a los que serán los usuarios finales de la aplicación para que nos ayuden a entender la visión real de lo que vamos a desarrollar. Podemos desglosar las fases del diseño en:

- Diseño conceptual. En esta fase hemos de concretar la respuesta a las siguientes preguntas. ¿Quién hace qué? ¿cómo lo hacen? ¿dónde lo hacen? ¿cuándo lo hacen? ¿por qué lo hacen?
- Diseño lógico. Las respuestas a las anteriores preguntas plantean unos requisitos. En esta etapa nos ocupamos de ellos y se establece lo que es necesario hacer.
- Diseño físico. Definimos la implementación del diseño lógico. Establecemos todas las interfaces y componentes de la aplicación y se realizan las especificaciones previas a la codificación.

CONSIDERACIONES PARA EL DISEÑO DE APLICACIONES C/S

Para conseguir buenas aplicaciones c/s es muy importante tener en cuenta las posibilidades que aparecen en cada etapa del diseño. A continuación se presentan algunas consideraciones para c/s, teniendo en cuenta cómo pueden afectar tanto la implementación como el desarrollo de la aplicación.

- *Elección de la herramienta de desarrollo*. En este punto se debe tener en cuenta si la herramienta permite o no la generación en las

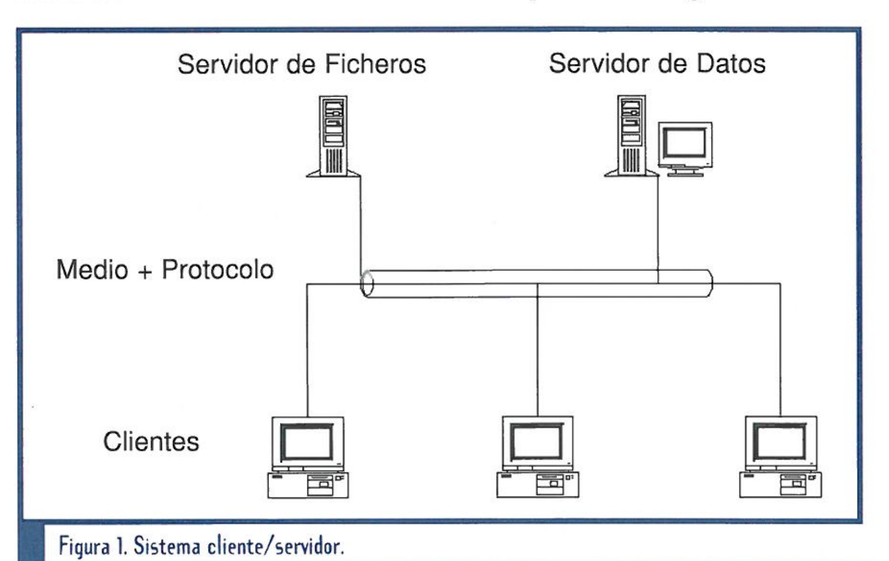


Figura 1. Sistema cliente/servidor.

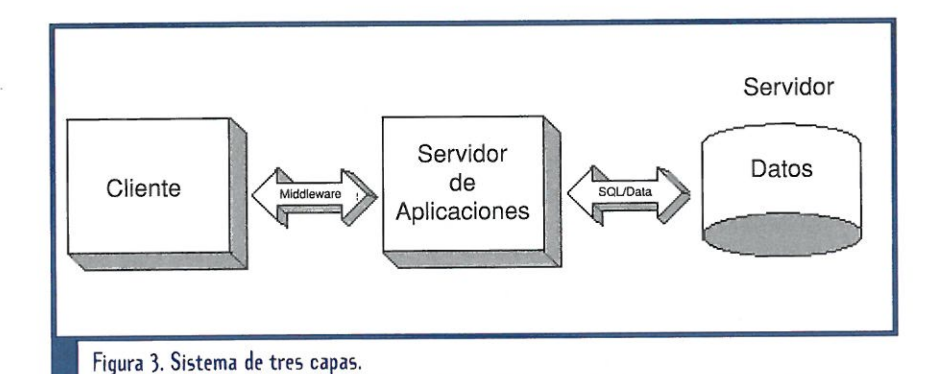


Figura 3. Sistema de tres capas.

datos figuran en una base de datos existente que es constantemente actualizada por diversas aplicaciones en explotación. La razón de almacenar los datos en diversas e incompatibles plataformas de información puede deberse a razones históricas, estratégicas o tecnológicas, o a que se trate de información de carácter confidencial a la que se desea que el acceso a los datos sea lo más restringido posible.

Podríamos desarrollar unas aplicaciones clientes que accediesen a los dos servidores de datos y se encargasen de la tarea, pero también podemos optar por otra solución que consiste en utilizar una aplicación que haga de puente entre las clientes y nuestro servidor de bases de datos. Esta aplicación también tendrá acceso al servidor con los datos de contratos. En un sistema de tres capas las aplicaciones clientes realizarán sus peticiones a la aplicación que se encuentra en la capa intermedia, que será la que se encargará de responder a las peticiones tratando con los dos servidores de bases de datos.

Existen numerosos productos orientados a la capa intermedia que soporten las diversas características de las bases de datos y solucionen muchos de los problemas de conectividad. El uso de estos productos puede reducir significativamente el tiempo y complejidad requeridos para construir las aplicaciones cliente/servidor robustas, escalables y muchas de ellas con varias capas.

MÉTODOS Y TECNOLOGÍAS DE ACCESO A DATOS

En el siguiente artículo desarrollaremos una aplicación cliente/servidor, pero como punto de entrada a continuación detallaremos diferentes métodos de acceso a un servidor de bases de datos.

SQL EMBEBIDO

El *SQL* embebido es un sistema mediante el cual se incrustan sentencias *SQL* en nuestros programas. Por ejemplo, *Oracle* dispone entre otros de *ProC* con el que podemos incluir en nuestro código fuente sentencias *SQL*. Una vez escrito el fuente se pasa por el precompilador de *ProC* lo que genera una salida de un programa en *C*. El código generado es críptico y dependiente del gestor de bases de datos. De este modo la potencia con la que se dota al programador es mayor pero perdemos portabilidad y facilidad en la programación.

ODBC

En consorcio de compañías de software lideradas por *Microsoft* fueron los desarrolladores de lo que conocemos hoy en día como la tecnología *ODBC*. Se trata de una *API*, una interfaz común programable mediante la que accedemos a nuestros gestores de bases de datos. Cada sistema de gestión de base de datos (*SGBD*) tiene *drivers* personalizados.

En un sistema de tres capas las peticiones se realizan a la aplicación intermedia

ODBC así como el resto de tecnologías que se detallan a continuación tienen en común una serie de características. La finalidad de los mismos es facilitar la tarea de programar contra una base de datos. Permiten utilizar sentencias *SQL* estándar, olvidándonos de las particularizaciones para cada gestor de bases de datos, lo que implica que la migración de un *SGBD* a otro no afectará a nuestras aplicaciones. Los elementos que forman la interfaz *ODBC* son:

- Una forma estándar de conectarse a *SGBD* para acceder a sus datos.
- Una sintaxis *SQL* basada en las especificaciones de *X/Open* y el *SQL* del *SAG* (1992).
- Un conjunto estándar de de datos.
- Biblioteca de funciones. Un conjunto de funciones *ODBC* mediante las cuales se permite el acceso a los datos de los diferentes *SGBD*.
- Códigos estándar de errores para su correcta interpretación.

Los beneficios para los desarrolladores son los siguientes:

- Las sentencias *SQL* pueden ser incluidas en el código literalmente o bien en tiempo de ejecución.
- El código de la aplicación es portable a cualquier *SGBD*.
- La aplicación es independiente del protocolo de red existente.

OLE DB

Es un conjunto de interfaces *COM* con el que se consigue un avance

considerable en la independencia "desarrollo de aplicaciones - almacenamiento de los datos". *OLE DB* permite trabajar con un amplio repertorio de tipos de datos independientemente del formato.

La finalidad de un sistema de tres capas es facilitar la tarea de programación contra una base de datos

Los componentes *OLE DB* son tres: proveedores de datos que publican los datos, consumidores de datos que los utilizan y componentes de servicio que los transportan y procesan.

- *Proveedores de datos.* Son los encargados de presentar los datos para su consumo.
- *Consumidores de datos.* El consumidor de datos *OLE DB* es cualquier aplicación que accede a los datos haciendo uso de la interfaz establecida, incluyendo herramientas de desarrollo, lenguajes.
- *Componentes de servicio de datos.* Ofrecen una funcionalidad sobre los datos, como los procesadores de consultas integración con *OLE DB*.

DAO (DATA ACCESS OBJECTS: OBJETOS DE ACCESO A DATOS)

La primera interfaz orientada a objetos para el motor de base de datos de *Access* (*Microsoft Jet*), permite usar *ODBC* para conectar a cualquier base de datos de la que poseamos su driver *ODBC*. *DAO* es apropiado para el desarrollo de aplicaciones dentro de un único sistema o para despliegues locales.

RDO (REMOTE DATA OBJECTS: OBJETOS DE ACCESO REMOTO)

Es una interfaz de acceso a datos para *ODBC* orientada a objetos. Se trata de una fina capa que pasa por la *API ODBC* y del administrador de controladores *ODBC*. Permite establecer las conexiones, crear cursores y ejecutar procedimientos usando muy pocos recursos en la estación de trabajo. Con *RDO* podemos limitar el número de registros que se procesan, ejecutar consultas de manipulación y definición de datos y utilizar procesamiento asíncrono controlado por eventos.

ADO (ACTIVEX DATA OBJECT: OBJETOS DE DATOS ACTIVEX)

Los objetos de datos *ActiveX* (*ADO*) permiten un acceso simplificado a toda la funcionalidad de *OLE DB*. *ADO* facilita un modelo de objetos sencillo con una interfaz única tanto para las bases de datos remotas como para las locales. Presenta un buen rendimiento y facilidad en la programación.

JDBC

JDBC es una *API* para implementar las comunicaciones con una base de datos mediante *Java*. Para trabajar con *JDBC* debemos disponer de un *driver* específico para la base de datos con la que trabajemos. Existen diferentes tipos:

- *Tipo 1*: Este *driver* trabaja realizando un puente entre dos tecnologías. Por ejemplo facilitándonos la conexión *JDBC-ODBC*. El *driver JDBC* traduce las llamadas *Java* a llamadas *ODBC* y devuelve el resultado a la aplicación.
- *Tipo 2*: Este *driver* realiza llamadas nativas a las bases de datos, genera llamadas nativas a la *API* de la *BBDD*. Este tipo de *driver* al igual que el anterior necesita realizar su instalación en la máquina que los va a utilizar, por lo que ninguno de ellos son una buena opción para implementar *applets* de conexión a datos en *Internet* ya que obligamos a realizar una instalación previa a la bajada se nuestro *applet*.
- *Tipo 3*: Este tipo de *driver* comunica al cliente con el servidor *sockets TCP/IP*. En el lado del servidor existe una aplicación residente que se encarga de transformar las peticiones de red en peticiones nativas. Este *driver* no necesita que sea instalado ningún código especial sobre el cliente.
- *Tipo 4*: *Java 100%*. El *driver* se comunica mediante *sockets TCP/IP* con el motor de la base de datos. Este tipo de *driver* es generalmente proporcionado por el vendedor de la misma. No necesita de ninguna instalación en la parte del cliente pero a diferencia del tipo anterior es dependiente de la *BBDD* utilizada.

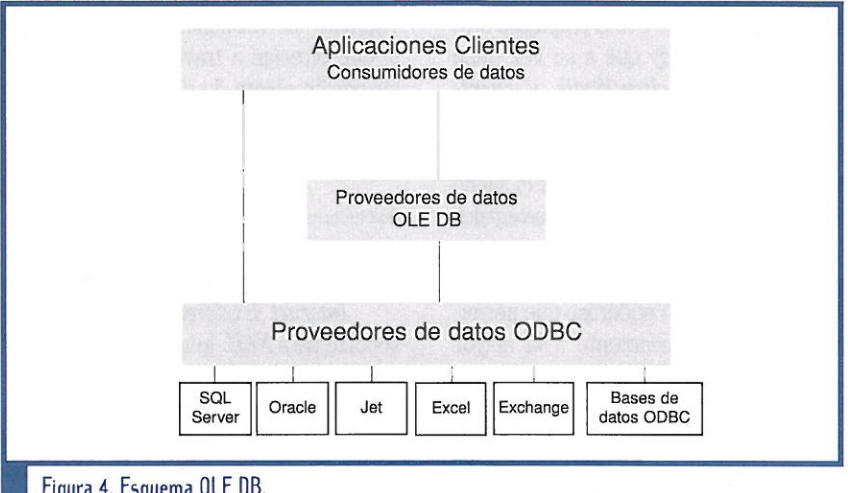


Figura 4. Esquema OLE DB.



XML (II). El DOM de XML

Adolfo Aladro García (aaladro@arrakis.es)

En este artículo nos adentraremos en el modelo *DOM (Document Object Model)* y aprenderemos a acceder y/o manipular la información que proporciona una fuente de datos en forma de documento *XML* dentro de una página *HTML*.

EL DOM (DOCUMENT OBJECT MODEL)

Todas aquellas personas que hayan desarrollado páginas *HTML* de cierta complejidad en cuanto a su faceta dinámica necesariamente habrán hecho uso de los objetos, propiedades y métodos que ofrece *DOM*. Es decir, *DOM* representa desde un punto de vista práctico la especificación que indica, por ejemplo, que el objeto *document* asociado a una página *HTML* tiene como propiedad otro objeto llamado *body* que a su vez tiene las propiedades *clientWidth* y *clientHeight*. De esta manera, si dentro de un *script* comprobamos el valor de *document.body.clientWidth* sabremos el ancho disponible de la ventana del navegador que contiene en ese momento nuestra página *HTML*. El *DOM* representa por lo tanto un recurso poderoso que permite acceder dinámicamente a la mayor parte de los elementos que conforman un documento *Web*.

A medida que el lenguaje *HTML* ha ido avanzando, paralelamente los navegadores han puesto a disposición del usuario un *DOM* más rico y completo, dotándoles de la facultad de consultar o modificar un número mayor de propiedades mediante lenguajes de *script* como *JavaScript*. Por ejemplo, podemos recordar que en la versión 3.x de *Internet Explorer* el *DOM* todavía no reconocía el objeto *image* dentro de una página *HTML*. Esto impedía que en ese navegador pudiera llevarse a cabo mediante *JavaScript* el típico efecto que hace que una imagen cambie cuando el ratón pasa por encima. Al carecer del objeto *image*, era imposible acceder a una imagen del documento para cambiarla. Por aquel entonces las versiones 3.x de *Navigator* sí que ofrecían a través de su *DOM* el susodicho objeto, lo que permitía escribir sentencias de código *Javascript* como la siguiente:

```
document.image[0].src =
    "img01.gif".
```

Internet Explorer 5.0 proporciona soporte para *XML* lo que significa, entre otras muchas cosas, que el *DOM* ha sido ampliado para poder manipular todos los aspectos relacionados con este estándar. En el artículo pasado vimos cómo era posible incrustar código *XML* de una página *Web* e incluso utilizamos el *DOM* cuando construimos el validador de documentos *XML*. Ahora entraremos en profundidad en los aspectos más importantes del *DOM*. Su conocimiento permitirá crear aplicaciones dinámicas que sepan tratar de forma adecuada los datos *XML*.

El DOM ha sido ampliado para poder manipular todos los aspectos relacionados con XML

Si bien vamos a centrarnos concretamente en el *DOM* proporcionado por *Internet Explorer 5.0*, éste se fundamenta con bastante fidelidad en el estándar propuesto por el *W3C*, lo que garantiza cierto grado de compatibilidad. El *DOM* implementado por *IE 5.0* es ciertamente algo complejo en tanto en cuanto abarca muchos aspectos relacionados con *XML*. Por el momento nosotros sólo nos detendremos en los más importantes dejando para más adelante los demás. El *DOM* de *XML* proporciona cuatro objetos base:

- El objeto *IDOMDocument*: representa el nodo a partir del cual se estructuran en forma de árbol todos los datos relacionados con el documento *XML*. Tiene todas las propiedades y métodos de un nodo (el objeto *IDOMNode*) y además cuenta con algunas que le son propias por tratarse del nodo superior.
- El objeto *IDOMNode*: representa un nodo dentro del árbol de datos. Los nodos serán los elementos principales a través de los cuales accederemos a los datos *XML*.
- El objeto *IDOMNodeList:* representa una colección de nodos (objetos *IDOMNode*).
- El objeto *IDOMNamedNodeMap:* representa una colección de nodos a la que podremos acceder mediante su nombre o por índice. Se suele utilizar típicamente para conocer los atributos asociados a una determinada etiqueta *XML*.

LOS DATOS EN FORMA DE ÁRBOL

La mejor forma que tenemos de conocer poco a poco el *DOM* de *XML* es partir de un documento *Web*, e ir viendo como podemos acceder a los distintos elementos que conforman una fuente de datos *XML* y saber cuáles son sus propiedades fundamentales. El listado 1 proporciona nuestro documento *XML* de partida. Se trata de la ya conocida fuente de datos con los discos de nuestra colección. Para poder experimentar con el *DOM* de *XML* hemos desarrollado una simple página *HTML* que carga los datos *XML* y que tiene un campo de texto donde podremos introducir una expresión para ser evaluada. El resultado de evaluar la expresión se mostrará dentro de la propia página *HTML*. La figura 1 muestra el aspecto de dicha página.

La forma que utilizamos para incluir los datos *XML* es la que ya vimos en el artículo anterior. La etiquetas *<XML>…</XML>* "rodean" al documento *XML* y el atributo *ID* proporciona el identificador para acceder a él. En nuestro caso concreto este atributo es *xmldoc* así que siempre que queramos evaluar una expresión referente a nuestros datos *XML* tendremos que empezar la misma por este atributo. Para evaluar la expresiones utilizamos la función *Javascript eval* la cual recibe como parámetro una cadena de texto que contiene la expresión que se va a evaluar.

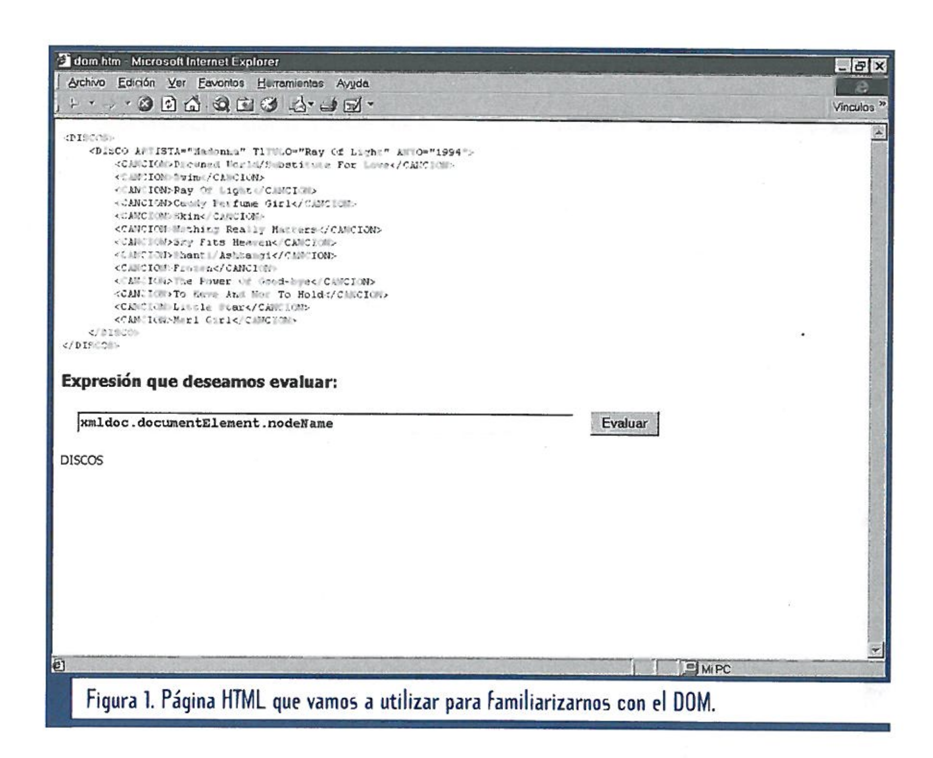


Figura 1. Página HTML que vamos a utilizar para familiarizarnos con el DOM.

Los nodos serán los elementos principales a través de los cuales accederemos a los datos XML

Como ya sabemos, los datos *XML* se organizan jerárquicamente en forma de árbol. Un árbol está formado por nodos que normalmente son hijos de algún otro nodo y que a su vez pueden ser padres de otros más. El estándar *XML* especifica que solamente puede existir un nodo raíz a partir de cual surjan todos los nodos hijos. La propiedad *documentElement* del documento *XML* embebido dentro de la página *HTML* contiene ese nodo raíz. Así tenemos que la expresión *xmldoc.documentElement* devuelve un objeto del tipo *IDOMNode* correspondiente a la raíz del árbol que representa a los datos *XML* o *NULL* si dicha raíz no existe. A partir de este nodo podremos explorar el árbol de datos.

A pesar de lo dicho anteriormente tenemos que poner de manifiesto que el objeto asociado a la referencia *xmldoc* es en sí mismo también un nodo, el que podríamos denominar nodo raíz de todo el documento *XML*. Por este motivo puede incluir además de los datos *XML* la información relativa a un *DTD*, por ejemplo. En otras palabras, todo lo relacionado con los datos *XML* está sujeto a ese nodo. Lo que hemos denominado nodo raíz se refiere exclusivamente a los datos *XML*. Veamos a continuación algunas de las propiedades de un nodom, las cuales son aplicables a cualquier nodo en general y por tanto también al nodo raíz en concreto.

La propiedad *nodeType* devuelve el tipo del nodo, el cual determinará los posibles valores que éste puede tomar

Listado 1. Fuente de datos XML que vamos a utilizar para nuestro ejemplo.

```
<DISCOS>
  <DISCO ARTISTA="Madonna" TITULO="Ray Of Light" ANYO="1994">
    <CANCION>Drowned World/Substitute For Love</CANCION>
    <CANCION>Swim</CANCION>
    <CANCION>Ray Of Light</CANCION>
    <CANCION>Candy Perfume Girl</CANCION>
    <CANCION>Skin</CANCION>
    <CANCION>Nothing Really Matters</CANCION>
    <CANCION>Sky Fits Heaven</CANCION>
    <CANCION>Shanti/Ashtangi</CANCION>
    <CANCION>Frozen</CANCION>
    <CANCION>The Power Of Good-bye</CANCION>
    <CANCION>To Have And Nor To Hold</CANCION>
    <CANCION>Little Star</CANCION>
    <CANCION>Merl Girl</CANCION>
  </DISCO>
</DISCOS>
```

así como la posibilidad de que pueda o no tener hijos. La tabla 1 muestra una lista de los posibles valores que puede devolvernos esta propiedad y sus respectivos significados. Volviendo a nuestro ejemplo, si evaluamos en nuestra página *HTML* la expresión *xmldoc .documentElement.nodeType* obtenemos el valor 1 que se corresponde a un nodo del tipo *NODE_ELEMENT*. Si evaluamos en nuestra página *HTML* la expresión *xmldoc.nodeType* obtenemos el tipo *NODE_DOCUMENT*. Posteriormente iremos viendo los otros posibles tipos que pueden aparecer dependiendo de las características de la fuente de datos *XML*. Como el valor que devuelve esta propiedad es un número entero hemos creado el siguiente *array* en *JavaScript* para ser capaces de devolver nombres nemónicos y no los números.

```
var nodeTypes = new
    Array("NODE_ELEMENT",
    "NODE_ATTRIBUTE",
    "NODE_TEXT", "NODE_CDATA_SEC-
    TION", NODE_ENTITY_REFEREN-
    CE", "NODE_ENTITY",
    "NODE_PROCESSING_INSTRUC-
    TION", "NODE_COMMENT",
    "NODE_DOCUMENT", "NODE_DOCU-
    MENT_TYPE",
    "NODE_DOCUMENT_FRAGMENT",
    "NODE_NOTATION");
```

Si *tmp* es el nombre de una variable que contiene el número correspondiente al valor de la propiedad *nodeType* en algún nodo, la expresión *nodeTypes[tmp-1]* tendrá el nombre vinculado a dicho valor.

La etiquetas <XML>...</XML> rodean al documento XML y el atributo ID proporciona el identificador

La propiedad *nodeName* devuelve el nombre del elemento, el cual depende a su vez del tipo de nodo. La tabla 2 muestra los posibles valores que puede tomar esta propiedad dependiendo del valor de la propiedad *nodeType*. Así cuando evaluamos la expresión *xmldoc.documentElement.nodeName* obtenemos el valor **DISCOS**. Si evaluamos en nuestra página *HTML* la expresión *xmldoc .nodeType* obtenemos el literal **"#document"**. La propiedad *nodeValue* devuelve el texto asociado al nodo. Al igual que ocurre con la propiedad anterior, ésta también depende del valor que tome la propiedad *nodeType*. La tabla 3 refleja estas diferencias.

Las propiedades *firstChild*, *lastChild* y *childNodes* permiten hacer la exploración necesaria en toda estructura en forma de árbol. Las dos primeras devuelven el primero y el último nodo hijo respectivamente. La última devuelve una lista con todos los nodos hijo (un objeto del tipo *IDOMNodeList*). De esta manera si evaluamos la expresión *xmldoc.documentElement.firstChild.nodeName* obtendremos como es de esperar la cadena **DISCO.** Nótese que los valores que devuelven estas propiedades dependen directamente del valor de la propiedad *nodeType*. Si tratamos de acceder al nodo hijo de un nodo que no puede tener nodos hijos obtendremos un valor **NULL** como resultado.

Las propiedades *nextSibling* y *previousSibling* del mismo modo que las anteriores nos permitían explorar los nodos hijo de un nodo dado, éstas hacen posible acceder a los nodos hermanos, concretamente el nodo hermano anterior y al siguiente.

La propiedad *parentNode* permite acceder al nodo padre. Tal y como ocurría al acceder a los nodos hijo, si tratamos de acceder al nodo padre de un nodo que no puede tener nodo padre obtendremos un valor **NULL** como resultado. Por ejemplo, la expresión *xmldoc.parentNode* devuelve **NULL** ya que el nodo que presenta la expresión *xmldoc* es un nodo de tipo *NODE_DOCUMENT* y este tipo de nodos no pueden aparecer como nodo hijo de ningún otro.

Por último la propiedad *ownerDocument,* dentro del grupo de propiedades mediante las cuales podemos explorar el árbol de datos a partir de un nodo dado, se encuentra esta propiedad. Por ejemplo, rizando el rizo podemos observar el valor de la expresión *xmldoc.documentElement .firstChild.ownerDocument.nodeName*.

Tabla 1. Valores que puede tomar la propiedad *nodeType*.

Valores posibles de nodeType	Significado
NODE_ELEMENT(1)	Representa un elemento. Puede tener como nodos hijo un nodo de alguno de los siguientes tipos:NODE_ELEMENT, NODE_TEXT, NODE_COMMENT, NODE_PROCESSING INSTRUCTION,NODE_CDATA_SECTION y NODE_ENTITY_REFERENCE. Puede ser hijo de uno de los siguientes nodos: NODE_DOCUMENT, NODE_DOCUMENT_FRAGMENT. NODE_ENTITY_REFERENCE y NODE_ELEMENT.
NODE_ATTRIBUTE(2)	Representa un atributo de un elemento. Puede tener como nodo hijo un nodo de alguno de los siguientes tipos: NODE_TEXT y NODE_ENTITY_REFERENCE. No puede ser hijo de ningún otro nodo.
NODE_TEXT(3)	Representa el texto contenido en una etiqueta. No puede terner nodos hijo. Puede aparecer como hijo de un nodo de uno de los siguiente nodos: NODE_ATTRIBUTE,NODE_DOCUMENT_FRAGMENT, NODE_ELEMENT, y NODE_ENTITY_REFERENCE.
NODE_CDATA_SECTION(4)	Representa un seccion CDATA. Las secciones CDATA se utilizan para "escapar" bloques de texto que de otro modo serían interpretados como código. No puede tener nodos hijo. Puede aparecer como hijo de uno de los siguientes nodos: NODE_DOCUMENT_FRAGMENT,NODE_ENTITY_REFERENCE, NODE_ELEMENT.
NODE_ENTITY_REFERENCE(5)	Representa una referencia a una entidad. Puede tener nodos hijo de uno de los siguientes tipos: NODE_ELEMENT, NODE_PROCESSING_INSTRUCTION, NODE_COMMENT, NODE_TEXT, NODE_CDATA_SECTION y NODE_ENTITY_REFERENCE. Puede aparecer como hijo de uno de los siguientes nodos: NODE_ATTRIBUTE, NODE_DOCUMENT_FRAGMENT, NODE_ELEMENT y NODE_ENTITY_REFERENCE.
NODE_ENTITY(6)	Representa a una entidad expandida. Puede tener como nodos hijo otras entidades expandidas. Puede aparecer como hijo de un nodo del tipo NODE_DOCUMENT_TYPE.
NODE_PROCESSING_INSTRUCTION(7)	Representa una instrucción de procesamiento. No puede tener nodos hijo. Puede aparecer como nodo hijo de un nodo de uno de los siguientes tipos: NODE_DOCUMENT, NODE_DOCUMENT_FRAGMENT, NODE_ELEMENT y NODE_ENTITY_REFERENCE.
NODE_COMMENT(8)	Representa un comentario. No puede tener nodos hijo. Puede aparecer como hijo de un nodo de uno de los siguientes tipos: NODE_DOCUMENT, NODE_ DOCUMENT_FRAGMENT, NODE_ELEMENT y NODE_ENTITY_REFERENCE.
NODE_DOCUMENT(9)	Representa a todo el documento. Puede tener como hijo un nodo de uno de los siguientes tipos: NODE_ELEMENT (uno como máximo),NODE_ PROCESSING_INSTRUCTION, NODE_COMMENT y NODE_DOCUMENT_TYPE. No puede aparecer como hijo de ningún otro nodo.
NODE_DOCUMENT_TYPE(10)	Representa una declaración <!DOCTYPE >. Puede tener como hijo un nodo de uno de los siguientes tipos: NODE_NOTATION y NODE_ENTITY. Puede aparecer como hijo de un nodo del tipo NODE_DOCUMENT.
NODE_DOCUMENT_FRAGMENT(11)	Representa un fragmento de documento. Un fragmento de documento vincula un nodo o un subárbol con un documento que no tiene porqué estar contenido dentro del documento. Puede tener como nodos hijo un nodo de los siguiente tipos: NODE_ELEMENT, NODE_PROCESSING_INSTRUCTION, NODE_COMMENT, NODE_TEXT, NODE_CDATA_SECTION y NODE_ENTITY_REFERENCE.
NODE_NOTATION(12)	Representa una notación dentro de una declaración <!DOCTYPE >. No puede tener nodos hijo. Puede aparecer como hijo de un nodo del tipo NODE_DOCUMENT_TYPE.

Primero accedemos al documento (*xmldoc*), que como ya hemos dicho es un nodo en sí mismo. Posteriormente accedemos al nodo raíz de los datos (*xmldoc.documentElement*). A partir de este nodo vemos cual es su primer nodo hijo (*xmldoc.documentElement .firstChild*) y finalmente accedemos al nodo raíz del árbol donde se encuentra este último nodo y comprobamos el nombre del mismo (*xmldoc.documentElement.firstChild.ownerDocument.node*

Name). El valor que obtenemos es evidentemente la cadena "#document". La expresión anterior es equivalente a *xmldoc.nodeName*. Este ejemplo es absurdo pero sólo se trata de que nos familiaricemos con la exploración del árbol asociado a un documento *XML*.

NodeType determinará la posibilidad de que un nodo pueda o no tener hijos

La propiedad *attributes* permite acceder a los atributos de una determinada etiqueta. En nuestro ejemplo concreto podemos acceder a los valores de los atributos *TITULO*, *ARTISTA* y *ANYO* de la etiqueta *DISCO* obteniendo primero una referencia al nodo correspondiente a la etiqueta *DISCO* y consultando posteriormente el valor de su propiedad *attributes*. La lista devuelve en un objeto del tipo *IDOMNamedNodeMap*.

LAS LISTAS DE NODOS

Tal y como hemos visto existen algunas propiedades que devuelven listas de nodos, por lo que se necesita un modo de recorrerlas. Casi todas las listas se manejan de la misma forma haciendo uso de *item* y de *length*. La siguiente función nos permitiría recorrer la lista de nodos:

```
function hijosNodo(nodo) {
  cadena = '';
  for(i=0;i<nodo.length;i++) {
    cadena += nodo.item(0).
    nodeName + '\n';
  }
  return cadena;
}
```

Teniendo en cuenta lo anterior si nosotros hacemos:

```
var nodo = xmldoc.documentEle-
    ment.firstChild.childNodes;
```

y llamamos a la función anterior

```
var resultado = hijosNodo(nodo);
```

obtendremos una cadena de texto en la que aparecerá trece veces la palabra **CANCION** (Las trece canciones del disco de nuestra fuente de datos *XML*). Del mismo modo podemos acceder a los atributos de una etiqueta dada. Por ejemplo, la expresión *xmldoc.documentElement.firstChild.attributes.item(0).nodeValue* devolverá como resultado la cadena de texto **Madonna.** La expresión *xmldoc.documentElement.firstChild.attributes.item(1).nodeValue* devolverá como resultado la cadena de texto **Ray Of Light** y la expresión *xmldoc.documentElement.firstChild.attributes.item(2).nodeValue* devolverá como resultado la cadena de texto **1994**, que se corresponden con los valores de los atributos.

En el caso concreto de que estemos tratando con un objeto de tipo *IDOMNamedNodeMap* (es decir, una lista de atributos), además del mecanismo anterior podemos utilizar el método *getNamedItem*. Éste devuelve el nodo (un objeto *IDOMNode*) correspondiente a un atributo de un determinado nombre. En definitiva este método viene a hacer algo similar al método *item* pero en vez de acceder a los nodos mediante su índice, lo hacemos utilizando el nombre del atributo. Por ejemplo, las dos expresiones siguientes devuelven el mismo valor, la cadena **Madonna**:

```
xmldoc.documentElement.
    FirstChild.attributes.
    item(0).nodeValue
xmldoc.documentElement.first-
    Child.attributes.

     getNamedItem("ARTISTA").
    nodeValue
```

Otro de los métodos que no hemos comentado hasta el momento y que puede resultar muy útil a la hora de recorrer las listas de nodos es el método *hasChildNodes* del objeto *IDOMNode*. Este método permite saber si un determinado nodo tiene nodos hijo o no.

FUNCIONES RECURSIVAS

Con todo lo visto hasta el momento ya podemos realizar funciones en *JavaScript* capaces de recorrer por completo determinadas ramas o subramas del árbol. El lenguaje *JavaScript* soporta la recursividad. Todas aquellas personas que hayan estudiado alguna vez la recursividad, sea cual sea el lenguaje de programación utilizado, han empezado viendo el ejemplo típico del número factorial. Bien, seamos fieles a esa vieja tradición y echemos un vistazo a la función *factorial* implementada con *JavaScript*.

```
function factorial(n) {
  if (n==0) {
    return 1;
  } else {
    return (n*factorial(n-1));
  }
}
```

Como se puede observar la forma de programar la recursividad es similar a la que utilizamos en lenguajes como *C*. Teniendo presente lo anterior podemos llegar a deducir fácilmente cómo habría de ser una función que recorra todos los nodos que dependen de un nodo determinado y devuelva una cadena de texto con los valores (la propiedad *nodeValue*) de todos esos nodos.

```
function recorrerArbol(nodo) {
  var cadena = '';
  var i;
  if (!nodo.hasChildNodes()) {
    return(cadena + nodo.
    nodeValue);
  } else {

    for(i=0;i<nodo.childNodes.len
```

```
      gth;i++) {
        cadena +=
      recorrerArbol(nodo.childNo-
      des.item(i)) + '\n';
      }
      return cadena;
   }
}
```

El método *hasChildNodes* nos ayuda a determinar si ya hemos explorado todas la subramas del árbol. En otras palabras, cuando devuelve **false** sabremos que nos hayamos en un nodo final. Si devuelve **true** estaremos en un nodo del que todavía "cuelgan" más nodos.

En ese caso utilizando la propiedad *childNodes* podremos recorrer todos esos nodos e ir profundizando en el árbol. Nótese que en este caso tan sólo se pretende obtener los valores de aquellos nodos que representan a una entidad no compleja (es decir, cuando tenemos algo del tipo *<NOMBRE> Madonna</NOMBRE>*).

El lenguaje JavaScript soporta recursividad

Dependiendo del tipo de exploración que deseemos realizar la función se modificará en un sentido u otro aunque en líneas generales la estructura será la misma. Así mismo puede ser interesante en determinados casos plantearse distintas formas de recorrer un árbol.

RECAPITULANDO

Hasta aquí hemos visto un resumen bastante completo de todos los aspectos del *DOM* de *XML* que permiten consultar los datos. El documento *XML* de ejemplo que hemos utilizado era sencillo y no tenía ningún *DTD* asociado.

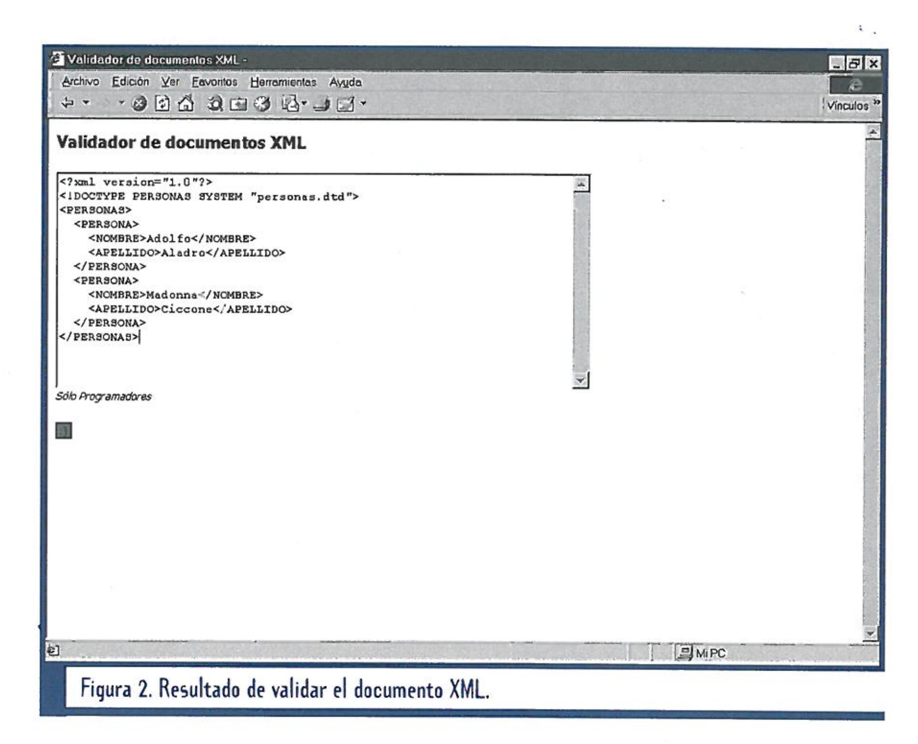


Figura 2. Resultado de validar el documento XML.

Ahora vamos a complicar un poco más este escenario introduciendo un documento *XML* que sí lo tiene y que además se carga en la página mediante el atributo *SRC* de la etiqueta *<XML>*.

Esto nos ayudará a comprender mucho mejor los tipos de nodo que hemos mencionado previamente así como su influencia sobre los posibles valores que estos pueden tomar.

Veamos en primer lugar este documento (el archivo **personas.xml**):

```
<?xml version="1.0"?>
<!DOCTYPE PERSONAS SYSTEM
    "personas.dtd">
<PERSONAS>
  <PERSONA>
    <NOMBRE>Adolfo</NOMBRE>
    <APELLIDO>Aladro</APELLIDO>
  </PERSONA>
  <PERSONA>
    <NOMBRE>Madonna</NOMBRE>
    <APELLIDO>Ciccone</APELLIDO>
  </PERSONA>
</PERSONAS>
```

Se trata de una sencilla base de datos en la cual se almacenan el nombre y el primer apellido de una serie de personas.

La primera línea del archivo indica que se trata de una fuente de datos *XML* y la segunda que se va a utilizar el archivo **personas.dtd** como validador, cuyo aspecto será el siguiente:

```
<!ELEMENT PERSONAS (PERSONA)+>
<!ELEMENT PERSONA (NOMBRE,
    APELLIDO)>
<!ELEMENT NOMBRE (#PCDATA)>
<!ELEMENT APELLIDO (#PCDATA)>
```

Un elemento de información denominado *PERSONAS* puede contener una o más ocurrencias del elemento de información denominado *PERSONA*. A su vez el ítem *PERSONA* estará siempre formado por dos elementos: *NOMBRE*" y *APELLIDO*.

Finalmente el elemento de información *NOMBRE* será literal y lo mismo ocurrirá con el ítem *APELLIDO*.

La figura 2 muestra el resultado que ofrece la página que hicimos en el artículo anterior cuando cargamos este documento *XML*. Una vez que la fuente de datos se carga en la página:

Tabla 2. Valores que puede tomar la propiedad *nodeName* dependiendo de los valores de la propiedad *nodeType*.

Propiedad nodeType	**Valor de la propiedad nodeName**
NODE_ATTRIBUTE	Nombre del atributo.
NODE_CDATA_SECTION	Literal "#cdata-section".
NODE_COMMENT	Literal "#comment".
NODE_DOCUMENT	Literal "#document".
NODE_DOCUMENT_TYPE	Nombre del tipo de documento. Por ejemplo,xxx en <!DOCTYPE xxx ...>.
NODE_DOCUMENT_FRAGMENT	Literal "#document-fragment".
NODE_ELEMENT	Nombre de la etiqueta.
NODE_ENTITY	Nombre de la entidad.
NODE_ENTITY_REFERENCE	Nombre de la entidad referenciada.
NODE_NOTATION	Nombre de la notación.
NODE_PROCESSING_INSTRUCTION	El primer token que sigue a los caracteres "<?"
NODE_TEXT	Literal "#text".

```
<XML ID="xmldoc"
    SRC="personas.xml"></XML>
```

podemos acceder al *DOM* para saber de qué manera se ha estructurado la información a partir de ese documento *XML*. Veamos en primer lugar cuantos nodos dependen del nodo superior de todo el árbol. Para ello procederemos a evaluar la siguiente expresión:

```
xmldoc.childNodes.length
```

El valor que obtenemos es **3**. Bien, ¿qué tres nodos son estos? Para verlo nada mejor que fijarnos en los tipos de cada uno de ellos, información que obtendremos evaluando :

```
xmldoc.childNodes.item(0).nodeType
xmldoc.childNodes.item(1).nodeType
xmldoc.childNodes.item(2).nodeType
```

Los valores que se obtienen son *NODE_PROCESSING_INSTRUCTION*, *NODE_DOCUMENT_TYPE* y *NODE_ELEMENT* respectivamente.

El primero de ellos se corresponde con la primera línea del archivo *XML*: *<?xml version="1.0"?>*.

Aquí aparece por primera vez un nodo que representa a lo que habíamos denominado del siguiente modo: "una instrucción de procesamiento".

El segundo de los nodos está asociado a la línea del documento *XML* donde se especifica el *DTD* que se va a utilizar para validar: *<!DOCTYPE PERSONAS SYSTEM "personas.dtd">*.

Por último, el tercero y último de los nodos es el que se corresponde con el nodo raíz de los datos, se trata por supuesto de *XML*.

Habíamos señalado que la propiedad *documentElement* también servía para acceder a este nodo raíz y por lo tanto podemos comprobar que las expresiones

Tabla 3. Valores que pueden tomar la propiedad *nodeValue* dependiendo de la propiedad *nodeType*.

Propiedad nodeType	**Valor de la propiedad nodeValue**
NODE_ATTRIBUTE	Cadena de texto que representa el valor del atributo. Para aquellos atributos que subnodos, será la concatenación de todas las cadenas de texto correspondientesa subnodos con entidades expandidas. Si se escribe este valor se borran todos los hijos del nodo y se reempla zan con un único nodo de texto que contiene el valor escrito.
NODE_CDATA_SECTION	Cadena que representa el texto almacenado en la sección CDATA.
NODE_COMMENT	Contenido del comentario.
NODE_DOCUMENT, NODE_DOCUMENT_TYPE, *NODE_DOCUMENT_FRAGMENT*, NODE_ELEMENT, NODE_ENTITY NODE_ENTITY_REFERENCE, NODE_NOTATION	Null.
NODE_PROCESSING_INSTRUCTION	Contenido de la instrucción de procesamiento excluyendo el target (que aparecerá en la propiedad nodeName).
NODE_TEXT	Cadena con el texto del nodo de texto.

```
xmldoc.documentElement.nodeName
xmldoc.childNodes.item(2).nodeName
```

son equivalentes y ambas devuelven el valor **PERSONAS**.

Al introducir un comentario aparece un nodo del tipo Node Comment

Si quisiéramos saber exactamente cual es el nombre del archivo *DTD* que se utiliza para validar el documento *XML* simplemente tendríamos:

```
xmldoc.childNodes.item(1).
    attributes.item(0).nodeValue
```

En primer lugar accedemos al nodo superior del árbol (*xmldoc*), luego al nodo correspondiente a su segundo hijo, el que se corresponde con la declaración del *DTD* (*xmldoc.childNodes .item(1)*), entonces consultamos el valor del único atributo que tiene y obtenemos **personas.dtd**.

Veamos cómo recorremos la subrama del árbol que contiene los datos *XML* (*xmldoc.childNodes.item(2)* o *xmldoc.documentElement*).

La tabla 4 muestra los resultados que obtenemos al explorar determinadas ramas del árbol partiendo del nodo raíz de los datos.

No debemos olvidar en ningún momento que dependiendo del tipo de nodo que estemos tratando en cada momento los valores de las propiedades *nodeValue* y *nodeName* tomarán unos u otros valores. Supongamos ahora que introducimos un comentario en nuestro documento *XML* de la siguiente manera:

```
...
<!—Esto es un comentario -->
</PERSONAS>
```

Tabla 4. Resultados de explorar ciertas ramas del árbol de datos asociado al documento XML.

Referencia	nodeType	nodeName	nodeValue
xmldoc.documentElement .childNodes.item(0)	NODE_ELEMENT	PERSONA	null
xmldoc.documentElement .childNodes.item(0) .childNodes.item(0)	NODE_ELEMENT	NOMBRE	null
xmldoc.documentElement .childNodes.item(0) .childNodes.item(0) .childNodes.item(0)	NODE_TEXT	#text	Adolfo

En este caso si evaluamos la expresión:

```
xmldoc.childNodes.item(2).childNo-
    des.item(2).nodeType
```

obtendríamos **NODE_COMMENT**. Para comprobar el contenido del comentario simplemente tendríamos que consultar:

```
xmldoc.childNodes.item(2).childNo-
    des.item(2).nodeValue
```

A lo largo de este artículo hemos introducido el *DOM* de *XML*. Todavía quedan muchos aspectos por analizar pero lo visto hasta el momento conforma el sustento indispensable y necesario para empezar a trabajar con nuestros datos *XML* desde los *scripts* de las páginas *HTML*.

La propiedad documentElement sirve para acceder al nodo raíz

Esto hará posible que en el futuro podamos integrar y manipular dinámicamente toda la información de la que dispongamos dentro de los documentos *Web*.

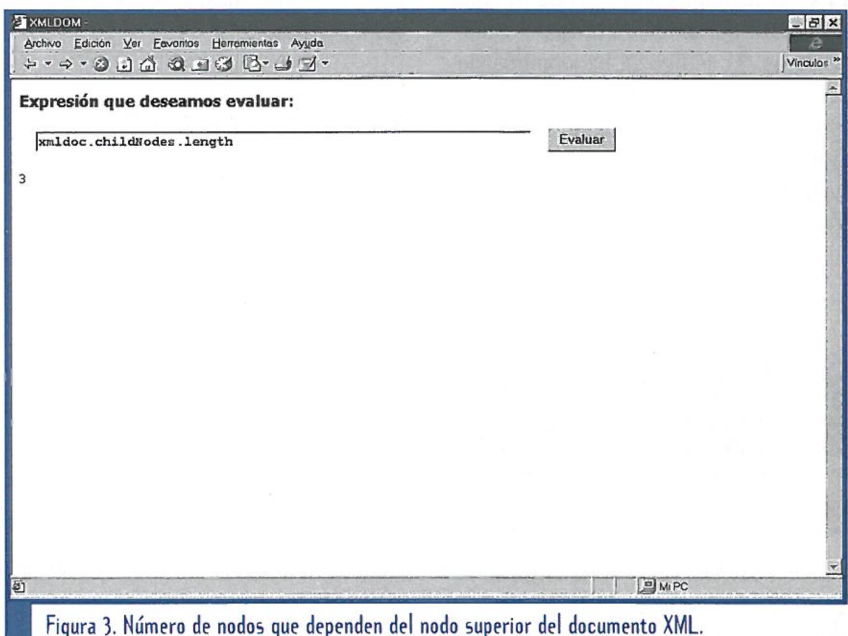


Figura 3. Número de nodos que dependen del nodo superior del documento XML.

Desarrollo de aplicaciones con videoconferencia (III)

Constantino Sánchez Ballesteros (constantino@nexo.es)

Ha llegado el momento de abordar la programación de *NetMeeting* utilizando el lenguaje *C++*, con el objetivo de aprender a manejar el canal de datos. Éste nos permite enviar y recibir información entre miembros de una conferencia.

UTILIZACIÓN DEL DATA CHANNEL

Nos vamos a basar en la aplicación *Chat* incluida en el *SDK* de *NetMeeting* para aprender a utilizar el canal de datos. El código se ha simplificado para resaltar las partes más importantes del programa. Mediante este programa podremos enviar y recibir mensajes utilizando el canal de datos de *NetMeeting*.

CREACIÓN DE UN CANAL DE DATOS

INICIALIZACIÓN

El objeto *INmManager* se sitúa en la parte más alta de la jerarquía de objetos *COM* de *NetMeeting*. Cualquier aplicación que utilice objetos *COM* de *NetMeeting* debe crear código que obtenga un puntero a un objeto de este tipo. Este puntero permite al programa crear sus propias llamadas y conferencias, detectar las propiedades del ordenador local y otras tareas de gran interés.

El canal de datos es el medio de transporte para audio, vídeo e información

Una vez que se ha obtenido un puntero a un objeto *INmManager*, conectaremos un sistema de notificaciones al mismo. Esto provocará que seamos avisados de la presencia de conferencias (*INmConferences*) y llamadas (*INmCalls*). El paso final consiste en inicializar el sistema *NetMeeting*, que será el encargado de lanzar la aplicación.

Nota: Es importante conectar un sistema de notificaciones a nuestro objeto *INmManager* antes de inicializar la aplicación *NetMeeting*. Si se cambia el orden de creación de estos pasos puede que perdamos notificaciones importantes.

En el siguiente código de ejemplo, el puntero a *INmManager* es inicializado sólo con capacidad para datos *(NMCH_DATA)*, ya que esta aplicación no está interesada en recibir notificaciones sobre otro tipo de tráfico (como audio o vídeo).

```
HRESULT InitManager(void) {
    HRESULT hr;
    hr =
    CoCreateInstance(CLSID_
    NmManager, NULL,

    CLSCTX_INPROC_SERVER,
    IID_INmManager,
         (void **)&g_
    pNmManager);
    if (SUCCEEDED(hr)) {
```

```
g_pManagerNotify = new
     CManagerNotify();
if (NULL != g_pManagerNotify ) {
hr = g_pManagerNotify->Connect(
     g_pNmManager );
if( SUCCEEDED( hr ) ) {
ULONG  uchCaps =  NMCH_DATA;
ULONG  uOptions =
     NM_INIT_NORMAL;
hr = g_pNmManager-
     >Initialize(&uOptions,
     &uchCaps)
               }
          }
     }
     return( hr );
}
```

- Método *Initialize.*

```
INmManager::Initialize
HRESULT Initialize(
     [in, out] ULONG * puOptions,
     [in, out] ULONG * puchCaps);
```

Activa el objeto *Manager* de conferencia y permite a la aplicación que efectúe la llamada registrarse por sí mismo con los servicios de conferencia. Devuelve uno de los valores de la tabla 1.

OBTENIENDO UN PUNTERO A INMCONFERENCE

El objeto *INmConference* permite acceder a canales y miembros (*INmChannels* y *InmMembers*). Una conferencia activa es aquella en la cual existen dos o más ordenadores (cada uno de ellos representado por un puntero *INmMember*), enviando y recibiendo audio, vídeo, ficheros o diferentes datos.

El objeto *INmManager* es obtenido de las conferencias existentes a través de su sistema de notificaciones. Cuando un puntero *INmManager* es inicializado por primera vez, o cada vez que se crea una conferencia, el objeto *COM* llama a nuestro método *INmManagerNotify:ConferenceCreated.*

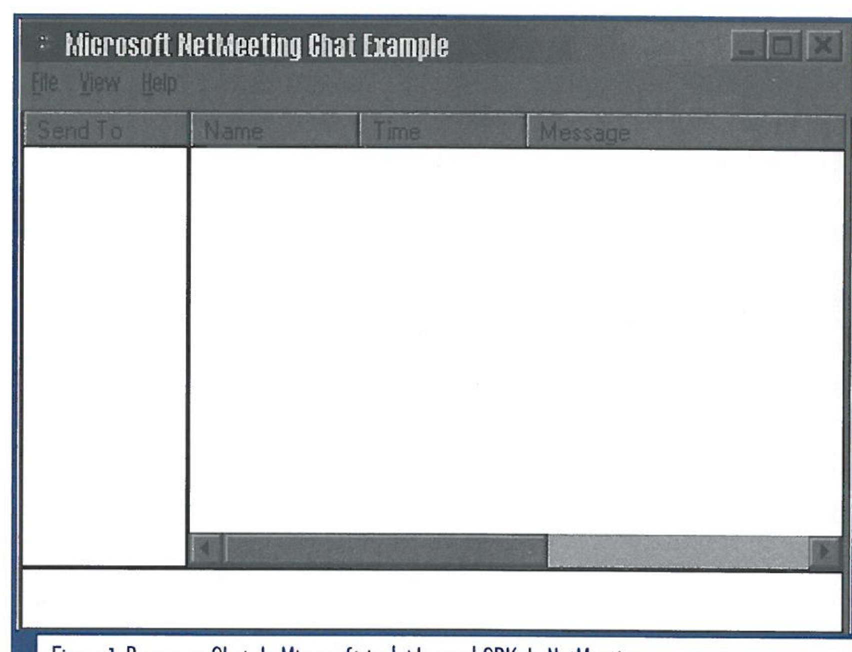


Figura 1. Programa Chat de Microsoft incluido en el SDK de NetMeeting.

Una conferencia se puede crear a través de la interfaz de usuario de *NetMeeting* (*UI*) o a través de llamadas a la *API COM* de *NetMeeting.*

El objeto INmManager se sitúa en la parte más alta de la jerarquía de objetos COM

El siguiente código es un ejemplo del método *ConferenceCreated* que tendremos que crear. Al igual que en la función precedente, crearemos un puntero al objeto Conferencia y conectaremos un objeto de notificación al puntero. Esto permitirá obtener notificaciones sobre la conferencia activa.

```
INmConference *g_pConference =
    NULL;
HRESULT STDMETHODCALLTYPE
 CManagerNotify::ConferenceCrea-
    ted(INmConference *pConferen-
    ce) {
    HRESULT hr = E_FAIL;

    pConference->AddRef();
    g_pConference = pConference;
    g_pConferenceNotify = new
    CConferenceNotify();

    if(NULL != g_pConference
    Notify) {
        hr = g_pConferenceNotify-
    >Connect( g_pConference );
    }
    return(hr);
}
```

CREACIÓN DE UNA INTERFAZ INMCHANNELDATA EN EL OBJETO INMCONFERENCE

INmChannelData es la interfaz con la que los datos específicos de una aplicación son transmitidos y recibidos. Un canal de datos se crea mediante un *GUID* (identificador global único). Los programas pueden usar el canal de datos para transmitir información sólo si ambos miembros han creado un canal de datos utilizando el mismo *GUID*. En el siguiente listado se crea un canal de datos para un programa que puede comunicarse con la aplicación *Chat* incluida en el *SDK* de *NetMeeting*. Si creamos primeramente un canal de datos, conectaremos al

Tabla 1. Método Initialize.

Valor	Significado
S_OK	Todo correcto.
E_FAIL	El objeto Manager de conferencia ya fue inicializado o los servicios de conferencia no están disponibles.
E_ACCESSDENIED	El objeto Manager de conferencia está siendo inicializado sobre otra línea sobre la que fue creada.

- puOptions. Puntero a ULONG que especifica las opciones que serán utilizadas. Pueden ser las siguientes:

NM_INIT_NORMAL	Inicia NetMeeting en estado normal. La interface de usuario de NetMeeting será visible.
NM_INIT_CONTROL	Requiere el control de la interfaz de usuario de NetMeeting. NetMeeting UI (interfaz de usuario) no será visible.
NM_INIT_NO_LAUNCH	No comienza NetMeeting si éste no está ejecutándose.

- puchCaps. Puntero a ULONG que especifica los tipos de canales permitidos. Pueden ser NMCH_ALL, NMCH_NONE o cualquier combinación de otros tipos.

NMCH_ALL	Todas las notificaciones de canales.
NMCH_AUDIO	Sólo notificaciones de conferencias de audio.
NMCH_DATA	Sólo notificaciones de conferencias de datos.
NMCH_FT	Sólo notificaciones de transferencias de archivos.
NMCH_NONE	Sin notificaciones de canales.
NMCH_SHARE	Sólo notificaciones de aplicaciones compartidas.
NMCH_VIDEO	Sólo notificaciones de conferencias de vídeo.

Si NM_INIT_NORMAL no se utilizó y NetMeeting estaba ejecutándose, el método devolverá el valor S_OK, pero cambiará puOptions a NM_INIT_NORMAL.
Si se utiliza NM_INIT_CONTROL y es devuelto por el método, entonces se deberá implementar el método NmUI, de tal forma que la aplicación deberá gestionar la visualización de todos los diálogos y otros elementos de la interfaz de usuario.
Cuando una aplicación acepta una llamada, la interfaz de usuario de NetMeeting se visualizará a menos que puOptions esté establecido como NM_INIT_CONTROL.

mismo un sistema de notificaciones para que las notificaciones del canal puedan ser recibidas.

```
// GUID de la aplicación local :
// {D29A8C50-774F-11d0-8B1D-
    00A0C91BC90E}
const GUID g_guidApp =
{ 0xd29a8c51, 0x774f, 0x11d0,
            { 0x8b, 0x1d, 0x0,
    0xa0, 0xc9, 0x1b, 0xc9, 0xe}
    };
```

```
INmChannelData *g_pChannelData =
    NULL;
HRESULT CreateChatChannel(void) {
    HRESULT hr;
```

Para poder crear el canal de datos utilizaremos el método *CreateDataChannel*. En él se deben especificar los parámetros correspondientes al puntero que representa el canal de datos y el *GUID* de la aplicación:

```
hr = g_pConference->
    CreateDataChannel
(&g_pChannelData, g_guidApp);
```

- Método *CreateDataChannel*.

```
HRESULT CreateDataChannel(
    [out] INmChannelData
    **ppChannel,
    [in] REFGUID rguid);
```

Provee un puntero a una interfaz *INmChannelData* sobre un nuevo objeto Canal. Este canal de datos, identificado por el *GUID* de nuestra aplicación, puede ser utilizado para transmitir información entre instancias de nuestra aplicación ejecutándose sobre cada ordenador de la conferencia. Devuelve uno de los valores que se ven en la tabla 2.

```
ppChannel. Puntero que apunta a
    otro puntero de la interfaz
    INmChannelData sobre el nuevo
    objeto Canal
rguid. GUID del canal de datos
```

Típicamente, una aplicación tendrá sólo un canal de datos y usará su propio *GUID* para identificar el canal.

De cualquier modo, una aplicación puede crear múltiples canales de datos e identificar cada uno con un *GUID* distinto.

El paso que se explica en el código que viene a continuación consiste en crear el sistema de notificaciones y enlazarlo al canal de datos mediante el método *Connect*:

```
if (S_OK == hr) {
g_pDataChanNotify = new
     CChannelDataNotify();

if(NULL != g_pDataChanNotify) {
hr = g_pDataChanNotify-
     >Connect(g_pChannelData);
     }
}
return hr;
}
```

ENVÍO Y RECEPCIÓN DE DATOS ESPECÍFICOS DE LA APLICACIÓN

El envío de datos se realiza a través del método *INmChannelData:SendData*. Los datos serán recibidos automáticamente con la ayuda del método *INmChannelDataNotify:DataReceived*.

Este método es llamado por el objeto *COM* de *NetMeeting* siempre que se reciban datos.

Una conferencia es activa cuando al menos dos equipos envían y reciben audio, vídeo o datos

En el siguiente listado se muestra cómo enviar y recibir cadenas de texto. El final de cada cadena termina con un carácter nulo.

```
HRESULT SendText(LPTSTR psz) {
 HRESULT hr;
 ULONG cb = lstrlen(psz);

 if ((0 == cb) || (NULL ==
     g_pChannelData)
     || (S_OK != g_pChannelData-
     >IsActive()) {
     return S_FALSE; // canal de
     datos no disponible
 }
```

El puntero **pMember** se refiere al integrante de la conferencia al que deseamos enviar el texto.

```
INmMember *pMember =
   GetSelectedMember();

cb++; // incluimos carácter nulo
   al final de la cadena de
   texto

hr = g_pChannelData-
   >SendData(pMember,cb,psz,0);
 return hr;

}
```

- Método *SendData*.

```
HRESULT SendData(
    [in] INmMember *pMember,
    [in] ULONG uSize,
    [in] BYTE *pvBuffer,
    [in] ULONG uOptions);
```

Envía un bloque de datos a través del canal a una aplicación específica sobre el miembro de la conferencia deseado. Ver valores en la tabla 3.

```
HRESULT STDMETHODCALLTYPE CChan-
    nelDataNotify::DataReceived
(INmMember *pMember, ULONG uSize,
    LPBYTE pb, ULONG dwFlags) {
LPTSTR psz;
psz = (LPTSTR) pb;
DisplayMsg(psz, pMember,
    dwFlags);
return S_OK;
}
```

Tabla 2. Método CreateDataChannel.

Valor	**Significado**
S_OK	Todo correcto.
E_POINTER	ppChannel es un puntero inválido.
E_INVALIDARG	El parámetro rguid contiene un valor inválido.
E_FAIL	El canal no puede ser creado, o recursos internos no están disponibles.
E_OUTOFMEMORY	No se pudo localizar memoria para el servicio requerido.

Tabla 3. Método SendData.

Valor	**Significado**
S_OK	Todo correcto.
E_POINTER	El puntero pvBuffer es inválido, o uOptions contiene una opción inválida.
E_NOINTERFACE	El objeto pMember no tiene la interface requerida.
E_FAIL	El buffer de datos especificado no fue enviado, el miembro especificado no está en la conferencia o recursos internos no están disponibles.
E_OUTOFMEMORY	No se pudo localizar memoria.
S_FALSE	pvBuffer es NULL o uSize es cero.

pMember. Puntero a la interface INmMember sobre el objeto Miembro al cual se enviarán datos. Si este parámetro es NULL los datos se enviarán a todos los miembros del canal.
uSize. Tamaño en bytes de los datos contenidos en el buffer.
pvBuffer.Puntero al buffer que contiene los datos que se enviarán.
uOptions. Flags para transferencia de datos. Reservado para usos futuros. Este flag debería establecerse como 0.

EJECUCIÓN REMOTA DE NUESTRO PROGRAMA

Para que el programa funcione es necesario que los demás usuarios tengan el mismo programa que nosotros. Pero, ¿cómo podemos asegurarnos que los integrantes de la conferencia tienen el mismo programa? A continuación veremos cómo obtener esta información.

El puntero pMember representa al miembro de la conferencia al que deseamos enviar el texto

La interfaz *COM* de *NetMeeting* permite a los programadores diseñar rápida y fácilmente *software* con capacidades de colaboración. Al utilizar esta interfaz los desarrolladores pueden crear aplicaciones que permitan a los usuarios trabajar conjuntamente sobre redes locales o remotas.

Para permitir este soporte, el *software* necesita asegurarse de que nuestra aplicación está ejecutándose sobre los demás equipos o intentar que los demás ordenadores lancen una copia de la aplicación.

Seguidamente veremos cómo lanzar copias de nuestros programas sobre equipos remotos. Esto se realiza principalmente con una llamada al método *INmConference:LaunchRemote*.

LIMITACIONES

Para lanzar una aplicación sobre un equipo remoto se deben tener en cuenta los siguientes detalles:

- La aplicación debe estar reconocida en el registro del ordenador remoto.
- La aplicación debe estar presente en el ordenador remoto.
- Tanto el ordenador remoto como el local deben estar utilizando *NetMeeting* o el objeto *COM NetMeeting*.
- Ambos sistemas deberían aparecer conjuntamente en una conferencia.

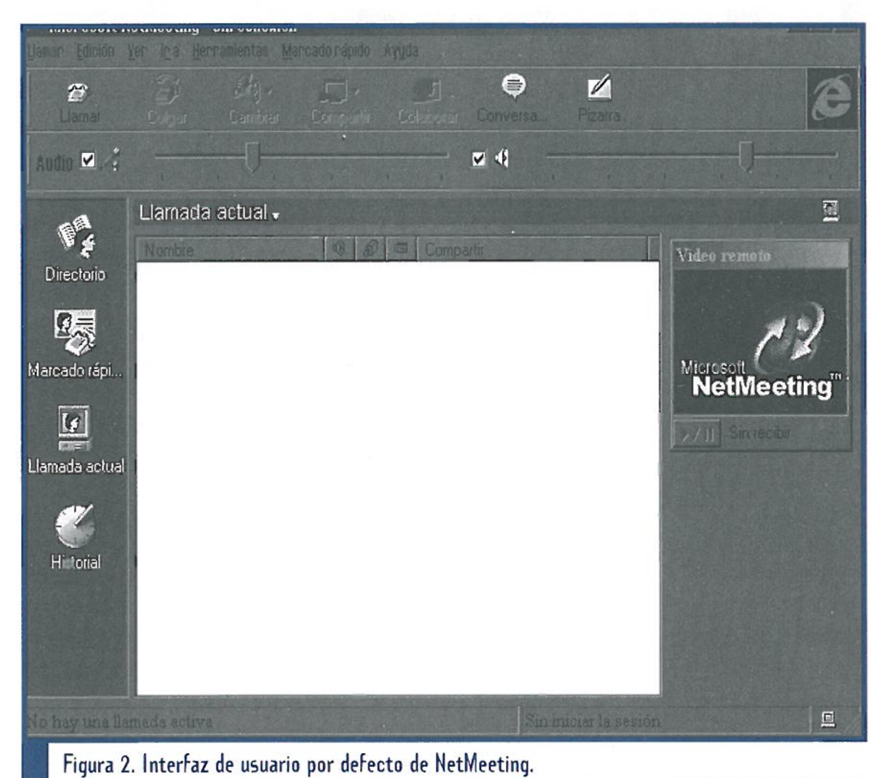


Figura 2. Interfaz de usuario por defecto de NetMeeting.

LAUNCHREMOTE Y EL REGISTRO

Las claves del registro para programas que pueden ser lanzados remotamente con *NetMeeting* se guardan en **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Conferencing\Applications.**

Bajo la clave de aplicaciones existe una clave para cada programa que puede ser lanzado remotamente. Esta carpeta debe ser nombrada con el *GUID* de nuestra aplicación.

La creación de un canal de datos implica que sólo se utilice audio, vídeo o datos de forma independiente

Las claves se describen en el siguiente apartado. El siguiente ejemplo muestra las claves del registro para el programa *Chat* de *NetMeeting*:

```
HKEY_LOCAL_MACHINE\
  SOFTWARE\
    Microsoft\
      Conferencing\
        Applications\
          {340F3A60-7067-11D0-
    A041-444553540000}
            Directory

    "C:\Archivos de programa\
    NetMeeting"
            Path
    "C:\ Archivos de programa
    \NetMeeting\cb32.exe"
```

ACTIVANDO Y OBTENIENDO INFORMACIÓN DE APLICACIONES REMOTAS

Las aplicaciones que permiten ser lanzadas remotamente se pueden introducir en el registro de varias formas. De todos modos, la mejor forma y más conveniente de registrar una aplicación es mediante el método *INmSysInfo:SetNmApp*. *SetNmApp* toma cuatro parámetros que son utilizados para registrar la aplicación en el sistema. La tabla 4 describe los parámetros que necesitamos incluir. El siguiente ejemplo muestra cómo utilizar el método *SetNmApp* para que nuestra aplicación agregue claves en el registro. También demuestra el uso del método *GetNmApp* para leer esos valores.

```
HRESULT hr;
INmManager *pManager;
INmSysInfo *pSysInfo;
// Insertamos código para obtener
     e inicializar pManager
// y pSysInfo
BSTR bstrApp, bstrCom, bstrDir;
LPTSTR_to_BSTR( &bstrApp,
     "C:\RemoteDemo\Demo.exe" );

LPTSTR_to_BSTR( &bstrCom,
     "C:\RemoteDemo\Demo Arg1
     Arg2" );
LPTSTR_to_BSTR( &bstrDir,
     "C:\RemoteDemo" );

hr = pSysInfo->SetNmApp(
          IID_APPGUID, // GUID pre-
     definido para esta aplicación
          bstrApp, // Ruta completa
     y nombre del archivo
          bstrCom, // Ruta y
     archivo con argumentos de
     línea de comandos
          bstrDir ); // Directorio
     de trabajo a utilizar
if( SUCCEEDED( hr ) ) {
     MessageBox( NULL, "Activada
     la aplicación como
     remotamente ejecutable ",
NULL, MB_OK );
} else {
     MessageBox( NULL, "No se pudo
     activar la aplicación para su
     uso remoto ",
                    NULL, MB_OK );
}

SysFreeString( bstrApp );
SysFreeString( bstrCom );
SysFreeString( bstrDir );
hr = pSysInfo->GetNmApp(
IID_APPGUID, // GUID para la
     aplicación sobre la que
     queremos saber

&bstrApp,      // Los 3 parámetros
     BSTR se rellenarán con la
     información
&bstrCom,      // obtenida del
     registro
&bstrDir );
if( SUCCEEDED( hr ) ) {
// Hacemos lo que sea con los
     parámetros BSTR que hemos
     obtenido
}
```

Tabla 4. Parámetros del método SetNmApp.

Parámetro	Uso/Significado	Clave del Registro
REFGUID rguid	Este GUID identifica la aplicación y es necesario cuando se llama a LaunchRemote Cada aplicación que vaya a ser lanzada remotamente debería tener un GUID que pueda ser utilizado para registrarlo con cualquier equipo.	En el registro, el valor de rguid es el nombre de la carpeta para la entrada a esta aplicación
BSTR bstrApplication	La ruta completa y el nombre de archivo del el módulo que será inicializado cuando se requiera para lanzar la aplicación asociada con el parámetro rguid. Si el parámetro bstrApplication es NULL, el nombre del módulo debe estar a partir del primer espacio en blanco tomado de la cadena bstrCommandLine.	Este parámetro es reflejado en el registro como la clave del registro Path bajo la carpeta rguid.
BSTR bstrCommandLine	Especifica la línea de comando a ejecutar. Si este parámetro es NULL, la función utilizará la cadena apuntada por bstrApplication como línea de comando.	Este parámetro es reflejado en el registro como la clave del registro CmdLine bajo la carpeta rguid.
BSTR bstrDirectory	Especifica la unidad y directorio para los procesos denominados hijos. La cadena debe contener una ruta completa y el nombre del archivo incluyendo la letra de la unidad.	Este parámetro es reflejado en el registro como la clave del registro Directory bajo la carpeta rguid.

- Método *GetNmApp*.

```
HRESULT GetNmApp(
    [in] REFGUID rguid,
    [out] BSTR *pbstrApplication,
    [out] BSTR *pbstrCommandLine,
    [out] BSTR *pbstrDirectory);
```

Encuentra los valores para una aplicación local de *NetMeeting* registrada. Devuelve uno de los valores que se ven en la tabla 5.

LANZANDO UNA APLICACIÓN REMOTA

Como ya se dijo anteriormente, las aplicaciones son lanzadas remotamente a través de una llamada al método *LaunchRemote*, el cual puede ser usado para iniciar una aplicación para un miembro específico de una conferencia o para todos los miembros.

LaunchRemote utiliza un *GUID* para saber qué aplicación lanzar. Dicha aplicación debe estar propiamente registrada (como ya sabemos) en el registro del ordenador remoto.

Puesto que una llamada a *LaunchRemote* puede enviar el mismo *GUID* es importante que registremos nuestra aplicación con el mismo *GUID*.

Para lanzar la aplicación a equipos remotos ésta debe estar presente en el ordenador remoto

A continuación se describen detalladamente los parámetros requeridos:

- *REFGUID rguid*: *rguid* es el *GUID* en el registro asociado con la aplicación que está siendo lanzada.
- *INmMember *pMember*: *pMember* es un puntero al miembro sobre el que se debe lanzar la aplicación. Si se utiliza el valor NULL *pMember* instruye a *NetMeeting* para que comience la aplicación sobre cada ordenador de la conferencia.

LaunchRemote retorna un resultado *HRESULT* que indica si el lanzamiento de la aplicación se realizó correctamente. Esto no garantiza una instancia remota de la aplicación que fue lanzada.

Tabla 5. Método GetNmApp.

Valor	**Significado**
S_OK	Operación efectuada.
E_OUTOFMEMORY	La función no pudo encontrar los parámetros.
E_POINTER	Los parámetros pbstrApplication, pbstrCommandLine, o pbstrDirectory son inválidos.

- rguid. GUID de la aplicación registrada.
- pbstrApplication. Puntero a una cadena que contiene la ruta completa y el nombre de archivo para la aplicación asociada con el parámetro rguid.
- pbstrCommandLine. Puntero a un BSTR que contiene la línea de comandos ejecutable que se pasará a la aplicación asociada con el parámetro rguid.
- pbstrDirectory. Puntero a un BSTR que contiene la unidad actual y el directorio para procesos "hijos". Esta cadena debe tener la ruta completa y el nombre de archivo incluyendo la letra.

- Método *LaunchRemote*.

```
HRESULT LaunchRemote(
    [in] REFGUID rguid,
    [in] INmMember *pMember);
```

Inicia la aplicación asociada con el *GUID* específico sobre el ordenador del miembro remoto de la conferencia.

Devuelve el valor *S_OK* si la operación se realizó satisfactoriamente o *E_POINTER* si *pMember* es un puntero inválido.

- *rguid*. *GUID* de la aplicación que llama.
- *pMember*. Puntero a la interfaz *INmMember* sobre el objeto remoto Miembro.

Si *pMember* es **NULL**, *NetMeeting* intenta comenzar la aplicación remota sobre todos los miembros en la conferencia.

En muchos de los casos, cada aplicación tendrá un canal de datos y el mismo *GUID* será utilizado para identificar el canal de datos y la propia aplicación.

En otros casos, una aplicación puede tener múltiples canales de datos, cada uno identificado con su propio *GUID*.

Un mismo GUID puede ser utilizado para identificar el canal de datos y la aplicación

En este caso, la aplicación utilizará uno de esos *GUIDs* para identificar su propio *GUID* de lanzamiento.

El siguiente código de ejemplo demuestra cómo utilizar *LaunchRemote* para lanzar nuestra aplicación sobre ordenadores remotos:

```
HRESULT hr;
INmConference *pConf;
IEnumNmMember *pEnum;
BOOL fLaunchOnEverybody;

     // Insertamos código para
     crear una conferencia con los
     miembros de pConf
     // ...


if (fLaunchOnEverybody) {

// Llamamos a LaunchRemote sobre
     todos los miembros de una
     conferencia

hr = pConf->LaunchRemote(
   IID_APPGUID, //
     GUID de la aplicación que
     va a ser lanzada

NULL );
} else
{
     // Llamamos a miembros en
     particular
     hr = pConf->EnumMember(
     &pEnum );

if( SUCCEDED( hr ) ) {
   INmMember *pMem;
   ULONG uFetched;

   while( SUCCEEDED( pEnum->
    Next( 1, &pMem, &uFetched ) {

if( IWantToLaunchOn( pMem ) ){

hr = LaunchRemote(

IID_APPGUID,pMem );

if( FAILED( hr ) ) {

// Hubo un problema en el
     lanzamiento
     }
  }
}
}

// no pude enumerar los miembros
}
```

CONCLUSIÓN

En el desarrollo de este tutorial hemos visto una variante para la comunicación entre varios equipos que desean realizar conferencias.

La solución más sencilla sería tener instalado en cada equipo la misma versión del programa y tenerla ejecutada para que no hubiera necesidad de andar con el registro y lanzar copias de nuestra aplicación sobre los equipos remotos.

Captura de secuencias con Delphi (I)

Juan Luis Ceada (i5939@fie.us.es)

Últimamente se han puesto de moda las tarjetas sintonizadoras de televisión. La mayoría de la gente las utiliza para entretenerse, aunque los programadores conseguiremos sacarles mucho más partido.

INTRODUCCIÓN

Las tarjetas sintonizadoras de *TV* se han vuelto muy populares, debido principalmente a la impresionante bajada de precios. Y en parte tenemos que agradecérselo a la empresa *Booktree*, que ha inundado el mercado con su chip *BT848* (y sucesores). De hecho, este chip es el que montan la mayoría de las tarjetas de gama baja (cuyos precios oscilan entre las 10.000 y 20.000 pesetas). Estas tarjetas permiten ver la televisión en el monitor del *PC*. Algunos modelos permiten incluso la recepción de teletexto (a una velocidad considerable) y de emisoras de radio *FM*. Todas incluyen, además, la posibilidad de capturar vídeo e imágenes. Si bien es cierto que la calidad del vídeo capturado no suele ser muy elevada, para nuestros propósitos será más que suficiente.

A lo largo de esta serie de artículos, crearemos un componente que permitirá capturar vídeo proveniente de diferentes fuentes (la señal de televisión, una cámara de vídeo, etc.). Comenzaremos implementando las capacidades básicas. En artículos posteriores, añadiremos gestores de eventos al componente, y algunas funciones nuevas. En el último artículo, crearemos una aplicación que permitirá enviar las imágenes capturadas a través de la red (ya sea una *intranet* o a través de *Internet*), simulando una video-conferencia.

¿POR QUÉ DELPHI?

¿Y por qué no? La creación del componente **JLCVideoPanel** surgió a raíz de que un cliente solicitase una aplicación destinada a controlar cámaras de vídeo (circuito cerrado). Básicamente, la aplicación debería poder mostrar en pantalla la imagen proveniente de las cámaras de vídeo que hubiese en el sistema de seguridad. También debería controlar, mediante comunicaciones por el puerto serie, diferentes parámetros de cada una de las cámaras, como el zoom, enfoque... aunque esto ya queda fuera del tema del artículo.

> Para poder probar el componente se necesita una tarjeta capturadora o sintonizadora

Por supuesto, el cliente quería una aplicación con un entorno gráfico amigable, y lo más urgentemente posible. Está claro que necesitábamos usar un lenguaje que nos permitiese desarrollar la interfaz de forma fácil y cómoda, y en el que pudiésemos crear nuestros propios componentes, de forma que pudiésemos reutilizar el código en futuros proyectos. *Delphi*

fue nuestra elección, y acertamos de pleno. Aunque usaremos a lo largo de los artículos *Delphi 4*, todo debería funcionar sin el más mínimo cambio en *Delphi 2* y *3*, versiones estándar o superior.

HARD & SOFT NECESARIOS

Con respecto al *hardware*, como habréis podido deducir por los párrafos anteriores, se hace necesaria la presencia en el sistema de una tarjeta capturadora de vídeo, o sintonizadora de *TV*. Aquellos que no posean dicho *hardware* podrán implementar el componente, pero evidentemente, no podrán verlo en funcionamiento. Aunque no es indispensable, seria conveniente que la *SVGA* pueda mostrar al menos 16.000 colores a 640x480. En próximos artículos veremos cómo grabar secuencias de vídeo, por lo que no estaría de más contar con un disco duro sobrado de "megas", y con un procesador medianamente potente.

Aunque a lo largo de los artículos utilizamos Delphi 4, todo debería funcionar perfectamente con las versiones 2 y 3

Con respecto al *software*, necesitaremos un sistema operativo *Windows* de *32 bits* (95, 95 *Osr2*, 98, *NT*), con las *DirectX* instaladas (versión 3.0 o superior). Por supuesto, a parte de las *DirectX* es necesario que los *drivers* de la tarjeta capturadora estén instalados. La mayoría de las tarjetas sintonizadoras que conozco instalan las *DirectX* a la vez que los *drivers*, así que supongo que siguiendo las instrucciones de instalación del manual de usuario nadie tendrá problemas relacionados con el *software*.

Tabla 1. Algunas funciones contenidas en AviCaptura.pas.

Función	Descripción
capDriverDisconnect	Desactiva el enlace entre el hardware de captura de vídeo y la ventana de visualización.
capDriverGetName	Obtiene el nombre del driver proporcionado por el fabricante de la capturadora.
capDriverGetVersion	Obtiene la versión del driver proporcionado por el fabricante de la capturadora.
capDlgVideoFormat	Muestra una ventana de selección de fuente de vídeo.
capDlgVideoSource	Muestra una ventana de selección de formato de vídeo.
capDlgVideoDisplay	Muestra una ventana de selección de display.
capDlgVideoCompression	Muestra una ventana de selección de compresión de vídeo.
capPreview	Activa el modo Preview.
capPreviewRate	Número de fps en modo Preview (visualización). No tiene nada que ver con el número de fps usado en la captura de vídeo.
capOverlay	Activa el modo Overlay.
CapPreviewScale	Activa el reajuste de tamaño automático. En modo Preview, el tamaño de los frames se adapta al de la ventana de visualización.

POSIBILIDADES DEL COMPONENTE

El componente puede dar mucho de si. Podría usarse en sistemas de seguridad. Por ejemplo, podríamos hacer que cuando saltase una determinada alarma, se comenzase a grabar vídeo, con una duración determinada. También podría usarse en reconocimiento de objetos por ordenador, teledetección, etc.

Hay que tener en cuenta lo siguiente: podemos capturar vídeo e imágenes procedentes de cualquier fuente. Una vez estén capturadas, es posible manipularlas a nuestro antojo, pudiendo desde aplicar un filtro para detección de bordes (para intentar reconocer objetos), hasta realizar una aplicación de vídeo-conferencia, enviando las imágenes capturadas a través de la red.

UN POCO DE TEORÍA

Es conveniente aclarar algunos conceptos a los que se hará referencia a lo largo de los artículos en multitud de ocasiones. Existen dos modos de visualizar la señal que llega a la tarjeta sintonizadora. El usuario puede cambiar de un modo a otro cuando lo desee.

- **Modo Preview**: en este caso, las imágenes que provienen de la capturadora de vídeo pasan a la memoria del sistema, y es el *GDI* de *Windows* el encargado de dibujar esas imágenes en pantalla. Por supuesto, este proceso consume tiempo de *CPU*. La visualización de imágenes puede reducirse si la aplicación pierde el foco, o si el sistema está muy cargado.
- **Modo Overlay**: en este caso, las imágenes que provienen de la

Campo	Descripción
WdeviceIndex	Numero de driver en el fichero SYSTEM.INI
FHasOverlay	¿El dispositivo puede realizar Overlay?
FHasDlgVideoSource	¿Puede mostrar el cuadro de selección de fuente de vídeo?
FHasDlgVideoFormat	¿Puede mostrar el cuadro de selección de formato de vídeo?
FHasDlgVideoDisplay	¿Puede mostrar el cuadro de selección de visualización de vídeo?
FCaptureInitialized	¿Está listo para realizar la captura?
FDriverSuppliesPalettes	¿El dispositivo puede crear paletas de color?
HVideoIn	No se usa en aplicaciones Win32
HVideoOut	No se usa en aplicaciones Win32
HvideoExtIn	No se usa en aplicaciones
HvideoExtOut	No se usa en aplicaciones Win32

Tabla 2. Estructura TCapDriversCaps. Indica las posibilidades del hardware a la hora de realizar capturas.

capturadora se dibujan directamente en la pantalla. Todo el proceso se realza por *hardware*, sin que el procesador intervenga en ningún momento.

Los dos modos son excluyentes. Es decir, la activación de uno implica la desactivación del otro. Otros términos comunes son:

- **Frame:** un *frame* es cada una de las imágenes que componen un vídeo. Es decir, un vídeo no es más que una sucesión de imágenes.
- **Frame rate:** indica el número de imágenes que se muestran/capturan por segundo. Se mide en *F.P.S* (*frames* por segundo)

El número de *F.P.S.* que capturar/visualizar sólo se tiene en cuenta cuando se está en modo *Preview*. En modo *Overlay* el número de *frames* coincide con el de la señal de vídeo, es decir, **25** *fps* o más.

Hay una cosa más a tener en cuenta: todas las capturas, tanto de vídeo como de imágenes individuales deben realizarse estando en modo *Preview*, pues de lo contrario no se realizará la captura.

LA BASE DE TODO: AVICAP32.DLL

Todo el proceso de captura/visualización de vídeo se realiza mediante llamadas a la librería *AVICAP32.DLL*, incluida en *Windows 95* y superiores. *AVICap* proporciona a la aplicación una interfaz simple, basada en el paso de mensajes que permite acceder al *hardware* de captura de vídeo y audio, así como controlar el proceso de captura de vídeo a disco. Esta librería proporciona multitud de métodos, desde los más básicos, para iniciar el *hardware* o grabar una imagen, hasta otros más complejos, destinados al control de dispositivos *MCI*. No usaremos todas las funciones disponibles durante el desarrollo del componente, y aún así el número de ellas resultará elevado.

Para trabajar más cómodamente desde *Delphi*, usaremos una unidad que encapsula la interfaz que ofrece *AVICap*, para que no nos tengamos que "pelear" con tipos de datos "raros" y punteros extraños. Incluye además la definición de diferentes mensajes del sistemas, así como constantes y tipos de datos. Se trata del fichero **AviCaptura.pas**, que podréis encontrar en el *CD-ROM* que viene con la revista. En la tabla 1 se muestran las funciones incluidas en dicha unidad que usaremos en la implementación del componente. No obstante, existen bastantes más, por lo que no estaría de más dar un repaso al código fuente, ya que algunas de ellas podrían resultar interesantes en determinados proyectos.

Todas las capturas, tanto de vídeo como de imágenes individuales deben realizarse en modo Preview

Existen tres estructuras de datos, que por su importancia, merecen ser comentadas. Se trata de *TCapDrivers-Caps*, *TcaptureParms* y *TCapStatus*. Por ahora sólo veremos la primera de ellas (tabla 2). Las otras dos se utilizan durante el proceso de captura de vídeo/imágenes, y las comentaremos en posteriores artículos. Existen algunas estructuras de datos más, pero su uso es menos común. Se encuentran definidas, como no, en **AviCaptura.pas**. Todo el proceso de visualización de imágenes y captura de vídeo depende del correcto uso que hagamos de estas funciones y estructuras. Aparentemente, todo esto parece muy complicado, aunque después veremos que nada más lejos de la realidad.

EL COMPONENTE

Hay que tener en cuenta varios aspectos antes de meternos de lleno en la programación del componente. En primer lugar, vamos a construir un componente Visual. Puesto que necesitamos un lugar donde poner la ventana de visualización y captura está claro que necesitamos un componente sobre el que se pueda dibujar, es decir, debe tener un *canvas*. Además, sería deseable que proporcionase una serie de propiedades y métodos, aparte de los propios

del proceso de captura, que nos permitan alinear, cambiar el tamaño, etc. del componente. Lo mejor y más cómodo es crear nuestro componente, mediante herencia, a partir de otro que proporciona *Delphi*: *TCustomPanel*.

Es importante no olvidar incluir las unidades AviCaptura y MMSystem en la cláusula Uses

TCustomPanel proporciona una serie de métodos y propiedades que serán de utilidad, y otras muchas que sobran, declaradas como *protected*. Haremos una selección, y las más interesantes las haremos públicas. Este es el motivo por el cual no heredamos de *TPanel*, ya que este hace públicas muchas propiedades de *TCustomPanel* que no vamos a necesitar para nada. Recordemos que es posible hacer que una propiedad protegida sea pública, pero no se puede realizar el proceso inverso. Es conveniente plantearse desde un principio cuáles son las propiedades y métodos que vamos a proporcionar al usuario. En la tabla 3 se enumeran las propiedades que vamos a implementar, mientras que en la tabla 4 tenemos los métodos. En próximos artículos ampliaremos el número de propiedades y métodos del componente. Por supuesto, no hay que dejar de lado el nombre de nuestro futuro componente, al que hemos denominado **JLCVideoPanel**.

DECLARANDO LA INTERFAZ DEL COMPONENTE

Antes de nada conviene aclarar que por problemas de espacio no es posible comentar el proceso de creación del componente en profundidad.

Tabla 3. Propiedades de JLCVideoPanel.

Nombre	Tipo	Acceso	Descripción
HandlePreview	THandle	L	Handle de ventana visualización
Activo	Boolean	L/E	Activar/desactivar componente
Overlay	Boolean	L/E	Activar/desactivar modo Overlay
FramesPreview	Dword	L/E	FPS durante la visualización
EscalaPreview	Boolean	L/E	Activa/desactiva ajuste automático de la señal de entrada al tamaño de la ventana de visualización
Align	--	L/E	Heredada de TCustomPanel. Hecha pública
Visible	Boolean	L/E	Heredada de TCustomPanel. Hecha pública
Enabled	Boolean	L/E	Heredada de TCustomPanel. Hecha pública
Dispositivo	Integer	L/E	Permite indicar que tarjeta sintonizadora usaremos Como normalmente sólo habrá una en el equipo, suele ser 0

Se supone que el lector tiene algo de experiencia en este tema. Basta con conocer lo básico: cómo crear propiedades, cómo leer y modificar el contenido de una propiedad, cómo declarar métodos, etc.

En caso de que no se tenga experiencia en el tema os recomiendo la lectura de algún libro (los propios manuales de *Delphi* suelen bastar). En *Internet* podréis encontrar un estupendo curso de creación de componentes en *Delphi*, en la dirección **personal .redestb.es/revueltaroche/**. Está en español, y para poder seguir el artículo con fluidez convendría leerse los cuatro primeros capítulos. En el CD-ROM que acompaña a la revista podréis encontrar todos los fuentes, con comentarios, que también os pueden resultar de ayuda.

Tabla 4. Métodos públicos de JLCVideoPanel.

Nombre	Descripción
SeleccionarFuente	Muestra el cuadro de diálogo de selección de la fuente de entrada
SeleccionarFormato	Muestra el cuadro de diálogo de selección del formato de visualización y captura en modo preview. Puede tener controles para activar la compresión por hardware
SeleccionarCompresion	Muestra el cuadro de diálogo de selección de formato de compresión (software)
SeleccionarDisplay	Muestra un cuadro de diálogo en el que el usuario puede activar o desactivar la opción de circuito cerrado. Ésta es una característica que sólo está disponible actualmente para los sistemas NTSC

Listado 1. Interfaz del componente.

```
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  ExtCtrls, MMSystem, AviCaptura;
type
  TJLCVideoPanel = class(TCustomPanel)
   private
    { Private declarations }
    FHandlePreview          : THandle;
    FVideoDriverNombre      : string;
    FActivo:boolean;
    FOverlay:Boolean;
    FFramesPreview:DWord;
    FEscalaPreview:Boolean;
    FDriverIniciado:Boolean;
    FDispositivo:Integer;

    function   AbrirDriver : Boolean;
    function   IniciarDriver(Index : Integer): Boolean;
    procedure  CerrarDriver;
    procedure  MostrarVideo;
    procedure  ActivarOverlay;
    procedure  ActivarPreview;
    procedure  SetActivo(Valor:boolean);  //procedimientos de
asignacion
    procedure  SetOverlay(Valor:boolean);
    procedure  SetEscalaPreview(Valor:Boolean);
    procedure  Conectar;
    procedure  Desconectar;
    function   TieneDlgFuente   : Boolean;
    function   TieneDlgFormato   : Boolean;
    function   TieneDlgDisplay   : Boolean;
  protected
    { Protected declarations }

  public
    { Public declarations }
    constructor Create(AOwner: TComponent); override;
    destructor  Destroy; override;
    procedure   Paint;override;
    procedure SeleccionarFuente;
    procedure SeleccionarFormato;
    procedure SeleccionarCompresion;
    procedure SeleccionarDisplay;
    {Propiedad publica, de solo lectura}
    property HandlePreview:THandle read FHandlePreview;
 published
    property Activo: boolean read FActivo write SetActivo
default false;
```

Lo primero es crear el nuevo componente. Para ello, usaremos la opción **New Component** del menú **Component**. Aparecerá un cuadro de diálogo en el que introduciremos **TCustomPanel** en el campo *Ancestor type*, **TJLCVideoPanel** en *Class Name* y **JLCSoft** (o cualquier otra cadena) en *Palette Page*. A continuación pulsaremos sobre el botón **Create Unit**. Con esta información *Delphi* creará el armazón del componente. A continuación tendremos que añadir las propiedades y métodos que aparecen en las tablas 3 y 4. Además, tendremos que incluir en la sección *private* las variables, funciones y procedimientos.

La modificación de las propiedades se realizará mediante procedimientos de asignación

Entre otras, en la sección *private* se definen las variables que almacenan el valor de las propiedades. También añadiremos unas cuantas funciones de uso interno. Además, es importante no olvidar incluir la unidades **AviCaptura** y **MMSystem** en la cláusula *uses*. Para asignar valores a las propiedades usaremos procedimientos de asignación. Es necesario hacerlo así, ya que cuando se cambia el valor de determinadas propiedades, se hacen algunas llamadas a la *API*, para actualizar el estado del componente. Esto nos permitirá, por ejemplo, poder pasar del modo *Overlay* al *Preview* sin tener que apagar y volver a activar la visualización de vídeo. Podemos ver un ejemplo en el listado 1. El valor de la propiedad *Activo* se guarda en la variable privada **FActivo**. Para modificar el contenido de dicha propiedad el componente llamará al procedimiento *SetActivo* que además de modificar el contenido de la variable hará que pasemos al modo *Overlay*. El resultado, después de unos minutos de trabajo, debe ser similar al que se aprecia en el listado 1.

En la sección *public* aparecen tres métodos a cuya definición acompaña la palabra reservada *override*. Dichos métodos son heredados del componente padre, pero nosotros los vamos a redefinir. Así, nuestro constructor se encargará de llamar al constructor del objeto padre, para a continuación, iniciar algunas variables del componente.

Delphi incluye un fichero de ayuda llamado Mm.hlp, en el cual encontramos información de la API Avicap32.dll

El destructor llamará al *Destroy* del objeto padre después de haberse asegurado de que el vídeo ha sido desactivado. Por último, nuestro método *paint* llamará al del padre, y a continuación se encargará de mantener el tamaño de la ventana de visualización idéntica al tamaño del panel. Si alguna vez desarrollamos un componente que descienda de **TJLCVideoPanel**, podríamos volver redefinir estos constructores y destructores a nuestro antojo. Los procedimientos y funciones que aparecen en la interfaz, pero no en la tabla 4 son de uso interno.

LA IMPLEMENTACIÓN

En la sección de implementación del componente se declaran dos nuevas variables, así como algunas constantes. Una de estas variables es **FAreaPreview**, que no es más que un puntero que indica sobre qué superficie pondremos la ventana de visualización. En este caso, dicha superficie será el propio componente, que recordemos, hereda de *TCustomPanel*, y que por lo tanto, al colocarlo en el formulario aparecerá con una forma similar a la del componente *Tpanel* (una superficie rectangular sobre la que el programador puede colocar otros controles). La variable **FVentanaPreview** apunta la ventana de visualización propiamente dicha. De hecho, al leer la propiedad **HandlePreview**, realmente

Listado 1. Continuación.

```
     property Overlay:Boolean read FOverlay write SetOverlay
default True;
     property FramesPreview:DWord read FFramesPreview write
FFramesPreview default 15;
     property EscalaPreview:Boolean read  FEscalaPreview write
SetEscalaPreview default True;
     property Align;
     property Visible;
     property Enabled;
```

Listado 2. Constructor, destructor y método paint del componente.

```
constructor TJLCVideoPanel.Create(AOwner: TComponent);
begin
  inherited Create(AOwner); {llamamos al metodo create de
TcustomPanel}
  parent:=AOwner as TWinControl;
  width:=260;       {ancho y alto inicial del componente}
  height:=180;
  FAreaVideo:=Self;          {
  FVentanaPreview:= 0;    {inicialmente, no hay ventana de
visualización}
  FHandlePreview:= 0;
  FVideoDriverNombre := 'No hay driver';
  FActivo:=False; {inicialmente, el componente no esta activo}
  FFramesPreview:=15;{por defecto, 15 fps en modo preview}
  FOverlay:=True;     {por defecto, modo preview activado}
  FEscalaPreview:=True;
  FDispositivo:=0;{por defecto, la primera tarjeta instalada}
end;

destructor TJLCVideoPanel.Destroy;
begin
  if FActivo then Desconectar; {si esta activo, desconectar}
  inherited Destroy;{llamar al destructor de TcustomPanel}
end;

procedure TJLCVideoPanel.Paint;
begin
  inherited Paint;{dibujar el componente}
// adaptar el tamaño de la ventana de visualización al del panel
  SetWindowPos(FVentanaPreview,HWND_TOP,0,0,
width,height,SWP_SHOWWINDOW);
end;
```

Listado 3. Iniciar y cerrar el driver.

```
procedure TJLCVideoPanel.SetActivo(Valor:boolean);
begin
     FActivo:=Valor;
     if Valor then     //Activar el vídeo
        Conectar
     else
        Desconectar
end;

procedure TJLCVideoPanel.Conectar;
begin
  if AbrirDriver then  //Intentamos activar el driver
  begin
    MostrarVideo;  //Comenzamos a visualizar imágenes, en modo
overlay o preview
    //dependiendo de lo que valga FOverlay
  end
  else
     raise Exception.Create('No se ha podido conectar el
video');
end;

procedure TJLCVideoPanel.Desconectar;
begin
     CerrarDriver;  //Cerramos el driver
     FActivo:=False; //Y ponemos FActivo a False
end;

function  TJLCVideoPanel.AbrirDriver : Boolean;
var
  achDeviceName    : array [0..80] of Char;
  achDeviceVersion : array [0..100] of Char;
begin
  // Crear la ventana de captura y visualización
  FVentanaPreview :=
capCreateCaptureWindow(PChar('JLCVideo'),WS_CHILD or WS_VISIBLE,
                              0, 0,FAreaVideo.Width,
FAreaVideo.Height,FAreaVideo.Handle, 0);
  //Si tenemos éxito en la creación....
  if FVentanaPreview <> 0 then
  begin
      //Damos valor a la propiedad HandlePreview, mediante su
variable privada
      FHandlePreview:=FVentanaPreview;
      //Abrir el driver de video del dispositivo Fdispositivo
(normalmente 0)
      FDriverIniciado := IniciarDriver( FDispositivo );
      //Si hemos conseguido conectar, obtenemos nombre y ver-
sión del driver
      if DriverStarted then
```

estamos obteniendo el contenido de **FVentanaPreview**. *Delphi* incluye un fichero de ayuda llamado **Mm.hlp**, en el cual encontraremos información de todas las funciones que usaremos en la implementación del componente (incluyendo parámetros de entrada, salida, etc.). Pasaremos ahora a comentar algunos aspectos concretos de la implementación.

CONSTRUCTOR, DESTRUCTOR Y PAINT

Vamos a redefinir estos tres métodos usando como base los que proporciona el componente *TCustomPanel.* El constructor se encarga de crear el componente y de asignar valores iniciales a las propiedades, mientras que el destructor desactivará el vídeo si está activo.

El método constructor se encarga de crear el componente y de asignar los valores iniciales

De esta forma, si hacemos que la propiedad *Aling* del componente tome el valor **alClient**, conseguiremos que al cambiar el tamaño de la ventana, cambie también el tamaño del componente, y con él, el de la ventana de visualización.

INICIAR Y CONECTAR EL DRIVER

Las siguientes funciones son usadas directa o indirectamente al activar la visualización de vídeo. Cuando el usuario coloca la propiedad *Activo* a **True**, se crea la ventana de visualización, usando la función *capCreateCap-*

tureWindow. A continuación, enlazamos dicha ventana al *driver* de la tarjeta sintonizadora (función *IniciarDriver, capDriverConnect(FVentanaPreview, Dispositivo)).* Cuando el usuario establece la propiedad *Activo* a **False**, se desconecta la ventana de visualización del *driver* (función *capDriverDisconnect (FVentanaPreview)*) y a continuación se destruye la ventana (*DestroyWindow(FVentanaPreview)*). Todo comienza cuando la propiedad *Activo* se modifica, ya que en ese momento empieza la ejecución del procedimiento *SetActivo*.

CAMBIAR EL MODO DE VISUALIZACIÓN

Antes vimos que al ejecutarse el procedimiento *SetActivo* si se conseguía abrir el *driver* (función *AbrirDriver*), se llamaba al procedimiento interno *MostrarVideo*. Éste, internamente llama a los procedimientos *ActivarOverlay* y *ActivarPreview*, según proceda. Como se puede ver en el siguiente código, para pasar de un modo a otro sólo hay que llamar a dos funciones, *capOverlay* y *capPreview*.

Cuando el usuario coloca la propiedad Activo a True se crea la ventana de visualización

De forma parecida, el procedimiento *SetEscalaPreview* activa el ajuste automático de las imágenes/vídeo capturadas al tamaño del componente (mediante interpolación). En este caso, se llama a la función *capPreviewScale(FVentanaPreview, VALOR),* donde **VALOR** puede valer **1** (activar) o **0** (desactivar). Consulte el código fuente del componente en el *CD-ROM.*

IMPLEMENTANDO LOS MÉTODOS PÚBLICOS

Mediante las funciones *SeleccionarFuente, SeleccionarFormato, SeleccionarCompresion, SeleccionarDisplay* podemos mostrar hasta cuatro cuadros de diálogo relacionados con la digitalización y captura de vídeo. El contenido, así como la posibilidad de mostrarlos o no, depende directamente de las capacidades de la tarjeta sintonizadora. Lo normal es que la tarjeta soporte la visualización de los cuatro. Puesto que el código es muy similar en los cuatro casos, sólo mostraremos el de uno de ellos. En concreto, veremos el del procedimiento *SeleccionarFormato*, puesto que el resto, al estar relacionados con la captura de vídeo, se comentarán en

Listado 3. Continuación.

```
      begin
//obtener el nombre y la version del driver que hemos instalado
if capGetDriverDescription(DriverIndex,achDeviceName,80,achDeviceVersion,100) then
      FVideoDriverNombre := string(achDeviceName);
    Result:=True;
  end
  else  //no hemos conseguido conectar, cerramos el driver
  begin
  Result := False;
  CerrarDriver;
  FVentanaPreview := 0;
 end;
end;

function TJLCVideoPanel.IniciarDriver(Index:Integer):Boolean;
begin
//intentamos enlazar el driver con la ventana de visualización que acabamos de //crear
     if capDriverConnect(FVentanaPreview, Index) <> 0 then
  Result := TRUE;
     else
         Raise Exception.Create('No se ha podido conectar el driver.');
end;

procedure TJLCVideoPanel.CerrarDriver;
begin
     //si hay ventana creada, la desconectamos del driver, y la destruimos
     if FVentanaPreview <> 0 then
     begin
       CapDriverDisconnect( FVentanaPreview );
       DestroyWindow( FVentanaPreview ) ;
       FVentanaPreview:=0;
      end;
end;
```


SÓLO PROGRAMADORES

Glide (y VI)

Constantino Sánchez Ballesteros (constantino@nexo.es)

Finalizamos esta interesante serie de artículos aprendiendo a gestionar los estados de *Glide*, la utilización de dos tarjetas gráficas, el uso de formatos de texturas y el trabajo con archivos gráficos con extensión 3DF.

G*lide* posee una colección de rutinas que devuelven información sobre el sistema, el *software* y la escena que está siendo renderizada. Trataremos diversos aspectos dentro de la obtención de información adicional por ejemplo sobre la configuración del sistema: la versión actual de *Glide*, número de *SST* presentes y tamaño del modo de vídeo, el cambio de la posición *Y* origen, el chequeo del estado de sistem, la utilización de dos tarjetas gráficas y el control del rendimiento del sistema.

INFORMACIÓN DE LA CONFIGURACIÓN

Mediante la función *grGlideGetVersion()* podemos saber con qué versión de *Glide* estamos trabajando.

```
void grGlideGetVersion(char version[80])
```

La función devolverá una cadena de caracteres que describirán la versión de *Glide* en la variable **version**, como por ejemplo **"Glide Version 2.2"**.

TAMAÑO DE LA PANTALLA

Las funciones *grSstScreenHeight()* y *grSstScreenWidth()* devuelven el ancho y alto en *pixels*, del *buffer SST*.

```
int grSstScreenHeight(void)
int grSstScreenWidth(void)
```

CAMBIANDO EL ORIGEN Y

La localización del origen *Y* se establece inicialmente como parte de la llamada a *grSstWinOpen()* en la secuencia de inicialización de *Glide*. Los valores iniciales pueden ser sobreescritos más tarde llamando a la función *grSstOrigin()*.

```
void grSstOrigin( GrOriginLocation_t origin )
```

El argumento *origin* especifica la dirección del eje de coordenadas *Y*. *GR_ORIGIN_UPPER_LEFT* coloca el origen de la pantalla en la esquina superior izquierda con valores positivos, mientras que *GR_ORIGIN_LOWER_LEFT* se sitúa en la esquina inferior izquierda con valor positivo.

CHEQUEANDO EL ESTADO DEL SISTEMA

Existen tres rutinas de *Glide* que nos ayudan a determinar el estado del *hardware* de la *Voodoo*.

```
void grSstIdle(void)
FxBool grSstIsBusy(void)
```

La primera bloquea hasta que el subsistema de la *Voodoo* esté inactivo. El sistema estará recargado cuando el hardware *FIFO* no esté vacío o cuando el motor esté sobrecargado. La segunda no efectúa bloqueo, sino que simplemente devuelve **FXTRUE** si el subsistema de la *Voodoo* está sobrecargado o **FXFALSE** en caso contrario. Podemos echar un vistazo a los contenidos del registro de estado llamando a *grSstStatus()*.

```
FxU32 grSstStatus( void )
```

Esta función devuelve un entero sin signo de 32 *bits* que contiene el estado del registro. Los bits obtenidos en este registro se definen en la tabla 1.

UTILIZANDO DOS TARJETAS GRÁFICAS

La función *grSstControlMode()* debería llamarse cuando se cambia entre la *VGA* y la tarjeta *Voodoo* para efectuar procesos como vídeo clips de introducción, etc. Es recomendable utilizar esta rutina en lugar de inicializar nuevamente *Glide*.

```
void grSstControlMode(GrSstCon-
     trolMode_t mode)
```

Con esta función se determina cuándo es visible la VGA y cuándo lo es la Voodoo, dependiendo del valor establecido en mode, el cual puede asumir uno de estos valores: *GR_CONTROL_ACTIVATE, GR_CONTROL_DEACTIVATE, GR_CONTROL_RESIZE, GR_CONTROL_MOVE.*

Los primeros dos valores se aplican a todos los sistemas. Cuando se especifica *GR_CONTROL _ACTIVATE*, el *Frame Buffer* de la *Voodoo* visualizará los gráficos en modo pantalla completa (*FullScreen*). Sobre sistemas *SST-96* se activará el bloque de vídeo. Si *mode* es *GR_CONTROL_DEACTIVATE*, se visualizará el *Frame Buffer* de la *VGA 2D*. *GR_CONTROL_RESIZE* es ignorado en modo *MSDOS*, *SST-1* y *SST-96*.

El argumento Origin especifica la dirección del eje de coordenadas Y

Para aplicaciones *Glide* en modo ventana, esta llamada redimensiona los *backbuffers* y *buffers* auxiliares, y es realizado por aplicaciones *Win32* en respuesta a mensajes *WM_SIZE*. La función *grSstControlMode()* puede fallar si no hay suficiente memoria de vídeo *Offscreen* para acomodar los *buffers* redimensionados.

GR_CONTROL_MOVE es ignorado bajo *DOS*, *SST-1* y *SST-96 FullScreen*. En aplicaciones modo ventana, esta llamada es utilizada para validar la localización y región de *Clip* asociada con el *FrontBuffer* cuando el usuario mueve una ventana, y es típicamente realizada por aplicaciones *Win32* en respuesta a mensajes *WM_MOVE*. Esto puede fallar si la implementación *DirectDraw* falla.

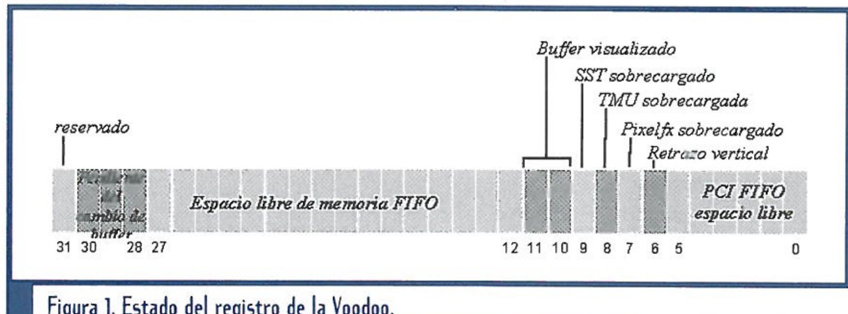


Figura 1. Estado del registro de la Voodoo.

Sobre sistemas *SST-1*, dado que los gráficos *2D* y *3D* existen sobre diferentes dispositivos, la activación o desactivación del *pass through* no requiere que volvamos a dibujar cada gráfico. Sobre sistemas *SST-96*, la aplicación es responsable del redibujado de los gráficos *2D* y *3D* cuando se utiliza *GR_CONTROL_ACTIVATE* ó *GR_CONTROL_DEACTIVATE*.

CONTROL DEL RENDIMIENTO DEL SISTEMA

El *hardware* de la *Voodoo* mantiene un *set* de cinco contadores que coleccionan estadísticas sobre el destino de los *pixels* cuando se mueven a través del *pixel pipeline*. *Glide* permite acceso a esos contadores a través de la esctructura *GrSstPerfStats_t* y la función *grSstPerfStats()*.

```
typedef struct GrSstPerfStats_s {

FxU32 pixelsIn;   /* # pixels pro-
     cesados  */
FxU32 chromaFail; /* # pixels no
     dibujados para chroma key */
FxU32 zFuncFail;  /* # pixels no
     dibujados para depth
     buffering */
FxU32 aFuncFail;       /* # pixels
     no dibujados para alpha */
FxU32 pixelsOut;  /* # pixels
     dibujados con vaciado de
     buffer */

} GrSstPerfStats_t;

void grSstPerfStats( GrSstPerfS-
     tats_t *pStats )
```

Tabla 1. Bits obtenidos en el registro.

Bit	Descripción
5:0	PCI FIFO espacio libre (0x3F=FIFO vacío)
6	Retrazo Vertical (0=activo; 1=inactivo)
7	Engine gráfico Pixelfx sobrecargado (0=engine parado; 1=engine sobrecargado)
8	TMU sobrecargada (0=engine parado; 1=engine sobrecargado)
9	Voodoo Graphics sobrecargada (0=parada; 1=sobrecargada)
11:10	Buffer visualizado (0=buffer 0; 1=buffer 1; 2=buffer auxiliar; 3=reservado)
27:12	Espacio libre de memoria FIFO (0xFFFF=FIFO vacía)
30:28	Número de comandos de buffers de intercambio pendientes
31	Interrupción PCI generada (esto no está implementado)

Para saber de cada *píxel* contado y guardado en *pixelsOut* se debe utilizar la siguiente ecuación:

```
pixelsOut = LfbWritePixels + buf-
    ferClearPixels +
(pixelsIn - zFuncFail - chroma-
    Fail - aFuncFail)
```

En *bufferClearPixels* se representa el número de *pixels* escritos como resultado de llamadas a *grBufferClear()* y puede calcularse así:

```
bufferClearPixels = (número de
    veces que el buffer fue
    vaciado) *
(ancho de clip window) * (alto de
    clip window)
```

La función *grSstPerfStats()* no espera a que el sistema se pare, y no incluye estadísticas para comandos que aún están en la *FIFO*. Por ello, es necesario llamar a *grSstIdle()* para vaciarla.

Todos los contadores son reseteados simpre que se llame a *grSstResetPerfStats()*. Los contadores hardware tienen un tamaño de 24 *bits*, por lo que debemos hacer llamadas regulares a *grSstResetPerfStats()* para evitar desbordes. El desbordamiento de contadores puede detectarse sin necesidad de llamar a *grSstResetPerfStats()*.

```
void grSstResetPerfStats(void)
```

FORMATOS DE LOS MAPAS DE TEXTURAS

La memoria de texturas es un recurso importante y limitado. *Glide* soporta una multitud de formatos de texturas para ayudar al programador a utilizar la memoria de las texturas fácilmente. Cada formato codifica la información del color por cada *texel* de diferente modo; muchos comprimen esta información de alguna forma. Los *texels* tienen 8 ó 16 *bits*, dependiendo del formato de textura, y se expanden a 32 *bits* antes de enviarse a la unidad de combinación.

Los formatos de texturas utilizan diferentes técnicas en la composición del color y la compresión del archivo

Glide utiliza nombres simbólicos para los formatos de texturas; el nombre describe la forma de codificar la información del color y la precisión. Por ejemplo:

- Los formatos de texturas *GR_TEXFMT_RGB_332* y *GR_TEXFMT_ARGB_8332* utilizan tres *bits* de color rojo y verde y dos bits para el color azul. Un valor de 8 *bits alpha* se incluye posteriormente.
- Los formatos de texturas *GR_TEXFMT_RGB_565*, *GR_TEXFMT_ARGB_1555* y *GR_TEXFMT_ARGB_4444* permiten tres formas diferentes de comprimir tres o cuatro componentes de color de 8 *bits* en 16 *bits*. El primer formato desecha el valor *alpha* y utiliza cinco bits para rojo y azul, y seis bits para el verde. El segundo utiliza cinco *bits* por cada valor de rojo, verde y azul, y guarda el *bit* extra para el valor *alpha*. El tercer formato trata los cuatro componentes de igual modo, utilizando cuatro bits por cada color.
- *GR_TEXFMT_INTENSITY_8*, *GR_TEXFMT_ALPHA_INTENSITY_44* y *GR_TEXFMT_ALPHA_INTENSITY_88* contienen un valor de intensidad en lugar de componentes de color y puede modelar efectos monocromos.
- El formato de textura *GR_TEXFMT_ALPHA_8* contiene sólo un valor de 8 *bits alpha*. Cuando el *texel* es expandido a un formato de 32 *bits ARGB*, el valor *alpha* es utilizado para rojo, verde y azul.
- *GR_TEXFMT_YIQ_422* y *GR_TEXFMT_AYIQ_8422* utilizan un canal de compresión para codificar la información del color. Cada *TMU* tiene almacenamiento para dos tablas de descompresión distintas que traducen la información codificada a valores de 32 *bits*.
- *GR_TEXFMT_P_8* y *GR_TEXFMT_AP_88* implementan una paleta de color. Cada *TMU* tiene espacio para una de las 256 entradas de color.

MEMORIA DE TEXTURAS

Cada *TMU* tiene su propia memoria para texturas, y tiene un rango en tamaño de 2 *Mb* a 4 *Mb*, dependiendo de la configuración del sistema. Para descargar una textura dentro de la memoria de texturas, se deben completar:

- Determinar cuánta memoria se requiere para la textura.
- Determinar la dirección de comienzo y extensión del espacio libre.
- Descargar la textura.
- Identificar la textura como la fuente de *texel* para las operaciones. *Glide* no maneja la memoria de texturas, pero incluye varias funciones que permiten a una aplicación manejarla correctamente.

DESCARGANDO MIPMAPS

Para descargar un *mipmap* en la memoria de texturas utilizaremos la función *grTexDownloadMipMap()*. Para reemplazar un nivel individual del *mipmap* llamaremos a *grTexDownloadMipMapLevel()*.

El primer argumento de estas rutinas es *TMU*, el cual designa la *TMU* como objetivo para la carga. Cada una de las rutinas tiene un argumento *startAddress* que especifica un *offset* (desplazamiento) dentro de la memoria de texturas donde la textura será cargada, y un argumento *evenOdd* que indica qué niveles cargar (especificado como *GR_MIPMAPLEVELMASK_EVEN*, *GR_MIPMAPLEVELMASK_ODD*, o *GR_MIPMAPLEVELMASK_BOTH*). *StartAddress* debe estar entre los valores retornados por *grTexMinAddress()* y *grTexMaxAddress()* y debe alinearse a 8 *bytes*.

La función *grTexDownloadMipMap()* espera los parámetros del *mipmap* en la estructura *GrTexInfo*; la otra rutina tiene argumentos para cada parámetro.

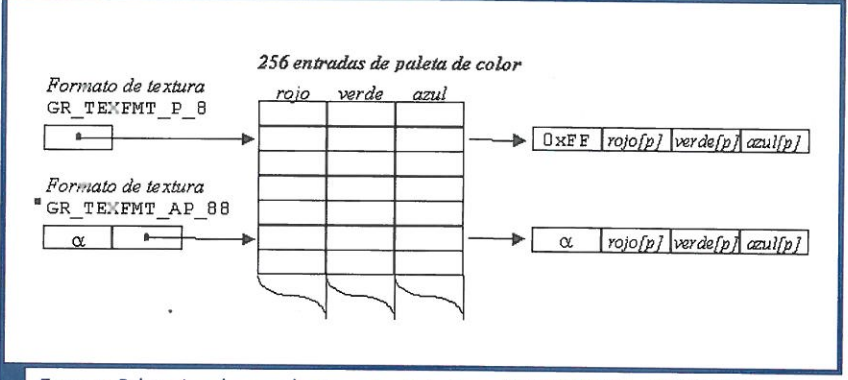


Figura 2. Paleta de color para los formatos de texturas GR_TEXFMT_P_8 y GR_TEXFMT_P_88.

DESCARGANDO TODO O PARTE DE UN MIPMAP

Como ya se ha comentado para cargar un *mipmap* debemos utilizar la función *grTexDownloadMipMap()*.

```
typedef struct {
    GrLOD_t        smallLod;
    GrLOD_t        largeLod;
    GrAspectRatio_t   aspectRa-
    tio;
    GrTextureFormat_t format;
    void   *data;
}    GrTexInfo;

void grTexDownloadMipMap(GrChi-
    pID_t tmu, FxU32 startAd-
    dress,

    FxU32 evenOdd, GrTexInfo
    *info )
```

La llamada a *grTexDownloadMipMap()* carga todos los *LODs* entre *GR_LOD_128* y *GR_LOD_8*. El segundo escenario carga sólo los *LODs* distintos. Volvemos a hacer la llamada para que la mayor dimensión de *LODs* sea múltiplo de 2. En ese caso, *GR_LOD_128*, *GR_LOD_32* y *GR_LOD_8* serán *LODs* distintos.

La información del sistema puede ser útil para saber si tenemos dos Voodoo conectadas en modo SLI

Debemos saber que no es necesaria ninguna modificación en los valores de la estructura *GrTexInfo* apuntados por **info**. *Glide* se saltará datos de la textura que contengan *LODs* distintos, cargando los que sean iguales.

CARGANDO UN MIPMAP EN MEMORIA FRAGMENTADA

Normalmente, los niveles de *mipmap* son guardados secuencialmente en la memoria de texturas. Cuatro direcciones base diferentes son especificadas para una textura multibase, cada una correspondiéndose con *GR_LOD_256*, *GR_LOD_128*, *GR_LOD _64* y una especial para texturas *GR_LOD_32* a través de *GR_LOD _1*. Para utilizar multibase debemos activarlo mediante una llamada a *grTexMultibase()*, descargar el *mipmap* como cuatro *mipmaps* más pequeños e iniciar el direccionamiento multibase mediante una llamada a *grTexMultibaseAddress()* cuatro veces con las cuatro direcciones de comienzo.

```
void grTexMultibase( GrChipID_t
    tmu, FxBool enable )
```

La función *grTexMultibase()* activa o desactiva el direccionamiento multibase. Debemos llamar a la función *grTexMultibaseAddress()* una vez por cada parte de una textura fragmentada con múltiples direccionamientos base.

```
void grTexMultibaseAddress(GrChi-
    pID_t tmu,
    GrTexBaseRange_t range,
    FxU32 startAddress,
    FxU32 evenOdd,
    GrTexInfo   *info
    )
```

El primer argumento de esta función nombra la *TMU* sobre la cual se cargará la textura fragmentada. *Range* declara a qué fragmento se está llamando y puede ser una de las cuatro:

- *GR_TEXBASE_256*
- *GR_TEXBASE_128*
- *GR_TEXBASE_64*
- *GR_TEXBASE_32_TO_1*

El tercer argumento, *startAddress*, es la dirección de comienzo para este

fragmento. Es necesario llamar a *grTexMultibaseAddress()* con una dirección de comienzo válida antes de que el fragmento sea descargado. El cuarto argumento, *evenOdd*, especifica que todas las texturas serán descargadas sobre esta *TMU*.

Un mipmap consta de una misma textura a diferentes resoluciones

En el siguiente ejemplo podemos ver el uso de registros de texturas multibase. Supongamos que el comienzo es una matriz de direcciones que han sido obtenidas a partir de una rutina de manejo de memoria. Supongamos también que el bloque de memoria de texturas que es apuntada por *start[0]* es suficientemente grande para *GR_LOD_256*, que el bloque apuntado por *start[1]* también lo es para *GR_LOD_128* y así sucesivamente. El *array* del *lod* guardará las cuatro constantes que identifican los fragmentos por conveniencia en el bucle *for* que inicializa los registros de base múltiple y descarga los fragmentos.

```
int i;
GrTexInfo info;
FxU32 start[4];
FxU16 mipmap[4][];
```

Establecemos rangos para el *LOD*:

```
GrTexBaseRange_t lod[4]=(GR_TEXBA-
     SE_256, GR_TEXBASE_128,
     GR_TEXBASE_64, GR_TEXBA-
     SE_32_TO_1);
```

Asignamos multibase de texturas para la unidad *TMU0*:

```
grTexMultibase(GR_TMU0, FX_TRUE);
for (i=0; i,4; i++) {
     info.smallLod = info.largeLod
     = lod[i];
     info.data = mipmap[i];
```

Determinamos las direcciones multibase y descargamos el *mipmap* mediante:

```
grTexDownloadMipMap:
grTexMultibaseAddress(GR_TMU0,
     lod[i], start[i],
          GR_MIPMAPLEVEL_BOTH,
     &info);
grTexDownloadMipMap(GR_TMU0,
     start[i],
          GR_MIPMAPLEVEL_BOTH,
     &info);
}
```

DESCARGANDO UNA TABLA DE DESCOMPRESIÓN O UNA PALETA DE COLOR

Los *texels* de los *mipmaps* que utilizan formatos de texturas *GR_TEXFMT_YIQ_422* y *GR_TEXFMT_AYIQ_8422* deben ser descomprimidos a valores de 32 *bits* antes de ser combinados y filtrados en la *TMU*. Los *texels* que son guardados en formatos de texturas *GR_TEXFMT_P_8* y *GR_TEXFMT_AP_88* deben ser previamente inicializados en una paleta de color para traducirlos a 32 *bits*.

En las paletas de color se guardan los detalles de los 256 colores utilizados en una textura

La tabla *NCC* o paleta de color debe ser descargada antes de que la textura utilizada pueda usarse como fuente para los *texels*. *Glide* tiene una rutina que puede descargar una paleta de color o una de las dos tablas de descompresión.

```
void grTexDownloadTable(GrChipID_t
     tmu, GrTexTable_t tableType,
     void *data)
```

La función *grTexDownloadTable()* descarga una tabla *NCC* o paleta de color a la *TMU*. El primer argumento nombra la *TMU* sobre la que se cargará la tabla. El segundo *tableType*, describe la clase de tabla que será descargada y se especifica con uno de los valores constantes: *GR_TEX_NCC0*, *GR_TEX_NCC1* o *GR_TEX_PALETTE*. El tercero apunta a los datos de la tabla, que deben ser del tipo *GuNccTable* ó *GuTexPalette*.

```
void
     grTexDownloadTablePartial(GrC
     hipID_t tmu, GrTexTable_t
     tableType,
void *data, int start, int end )
```

TABLAS DE DESCOMPRESIÓN

Una textura comprimida como textura *YAB* puede descomprimirse en la tabla de descompresión apropiada mediante la ayuda del programa **TexUS** incluido en el *SDK* de *Glide*. La textura comprimida es guardada con la extensión *3DF* y puede ser cargada utilizando la rutina de *Glide gu3dfLoad()*. *Glide* representa las tablas de descompresión *NCC* con la estructura de datos *GuNccTable*, mostrada a continuación.

```
typedef struct {
     FxU8  yRGB[16];
     FxI16 iRGB[4][3];
     FxI16 qRGB[4][3];
     FxU32 packed_data[12];
}    GuNccTable;
```

Antes de que una textura comprimida pueda ser utilizada como fuente de *texels*, una de las dos tablas *NCC* debe asignarse como fuente para las operaciones de descompresión. La función *Glide grTexNCCTable()* debería llamarse antes de cualquier operación de renderizado usando la tabla comprimida que se ha iniciado.

```
void grTexNCCTable( GrChipID_t
     tmu, GrNCCTable_t table )
```

La función *grTexNCCTable()* selecciona una de las dos tablas *NCC* sobre *tmu* como la fuente actual para las operaciones de descompresión. Los valores válidos son *GR_TEXTABLE_NCC0* y *GR_TEXTABLE_NCC1*. En el siguiente ejemplo vamos a ver la construcción de una rutina de carga de una tabla *NCC*. Las tablas *NCC* son creadas por programas en la librería *TexUS* y escritas a un fichero con formato *3DF*. Este segmento de código utiliza la función *gu3dfLoad()* para leer el fichero en memoria.

```
Gu3dfInfo info;
```

Descargamos la tabla incluida en el archivo gráfico **"ncctable.3df"**:

```
gu3dfLoad("ncctable.3df", &info);
grTexDownloadTable(GR_TMU0,
     GU_TEX_NCC1, &info.table.ncc-
     Table);
grTexNCCTable(GR_TMU0, GR_TEXTA-
     BLE_NCC1);
```

PALETAS DE COLOR

Una paleta de color es un *array* de 256 colores en formato *ARGB*, 8 *bits* por cada componente y 32 *bits* por entrada. La paleta se define utilizando la estructura *GuTexPalette*,

```
typedef struct {
     FxU32  data[256];
}     GuTexPalette;
```

Crearemos una paleta de color al azar y la descargaremos en la *TMU0*. Descargamos una textura paletizada y configuramos la textura y la unidad de combinación de color apropiadamente.

```
extern unsigned long lrand(
     void);
GuTexPalette palette;
int i, j;
// creamos una paleta de color al
     azar de 256 entradas
for (i=0; i<256; i++)
     palette.data[i] = 0x00FFFFFF
     & lrand();
grTexDownloadTable(GR_TMU0,
     GU_TEX_PALETTE, &palette);
```

CARGANDO MIPMAPS DESDE DISCO

El programa **TexUS** (*3Dfx Interactive's Texture Utility Software*) crea archivos en formato *3DF*. Estos archivos pueden contener *mipmaps*, tablas de descompresión o ambos. Con un par de tipos de datos diferentes y un par de funciones tendremos acceso a los ficheros *3DF*.

Las estructuras de datos se muestran más abajo. *Gu3dfInfo* es el nivel alto de la estructura.

```
typedef struct {
     FxU32  width, height;
     int    small_lod, large_lod;
     GrAspectRatio_t
     aspect_ratio;
     GrTextureFormat_t format;
}    Gu3dfHeader;

typedef union {
     GuNccTable  nccTable;
     GuTexPalettepalette;
} GuTexTable;

typedef struct {
     Gu3dfHeader header;
     GuTexTable  table;
     void    *data;
     FxU32  mem_required;
}    Gu3dfInfo;
```

La aplicación llama primeramente a la función *gu3dfGetInfo()* para rellenar datos en la estructura *Gu3dfInfo* apuntada por **info**.

```
FxBool gu3dfGetInfo( const char
     *filename, Gu3dfInfo *info )
```

Después de que una aplicación ha determinado con las características de un *mipmap* 3DF se debe localizar memoria para el mismo y su dirección que está guardada en el puntero **infoÆdata.**

```
FxBool gu3dfLoad( const char
     *filename, Gu3dfInfo *info )
```

Tanto *gu3dfGetInfo()* como *gu3dfLoad()* retornan el valor **FXTRUE** si el fichero especificado por *filename* existe y puede ser leido; en el caso contrario se retornará **FXFALSE.**

Cada TMU tiene su propia memoria para texturas

En el siguiente ejemplo se presenta la carga de un *mipmap*. El código llama a *gu3dfGetInfo()* para determinar los requerimientos de memoria, localiza espacio para el *mipmap* y lo lee.

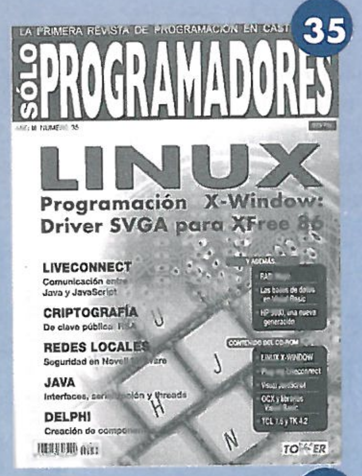
```
Gu3dfInfo fileInfo;
gu3dfGetInfo("mipmap.3df",
     &fileInfo);
fileInfo.data =
     malloc(fileInfo.mem_requi-
     red);
gu3dfLoad("mipmap.3df", &fileIn-
     fo);
```

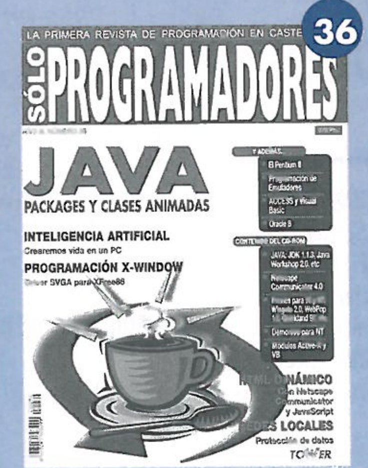
Con la teoría y práctica aquí mostrada el programador podrá comenzar a un nivel básico a crear mundos virtuales haciendo un uso extenso de la tecnología más vanguardista actualmente en el terreno de los gráficos *3D* para la plataforma *PC*. Para obtener nuevas actualizaciones de *Glide* y su *SDK* existe una dirección en *Internet*, **www.3dfx.com.** A lo largo de toda esta serie de artículos hemos realizado un acercamiento a las librerías *Glide* y sus tarjetas *3DFX Voodoo*.

números atrasados

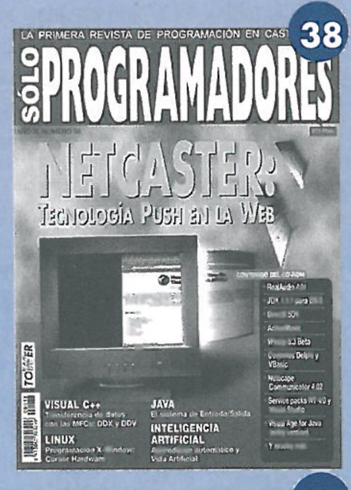
SÓLO PROGRAMADORES

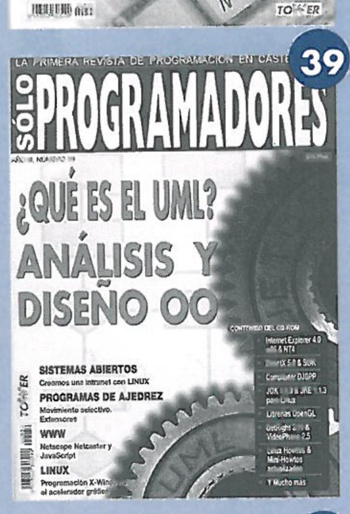
completa ya tu colección









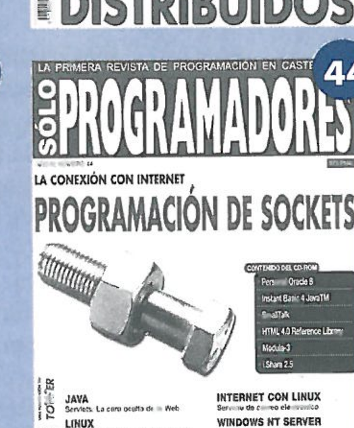
















SÓLO PROGRAMADORES

☐ **Sí, deseo suscribirme a la revista SÓLO PROGRAMADORES durante un año (12 números) eligiendo la siguiente modalidad.** (Marcar con una X)

☐ Suscripción **NORMAL**
9.500 ptas. / 57,10 €

☐ Suscripción **ESTUDIANTES**
7.600 ptas. / 45,68 €

Nombre ....Apellidos ....

Domicilio: calle ....Nº ....Piso ....

C.P: ....Población:..Provincia: ....

Teléfono: ....Fax: ....E-mail: ....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº: .... Fecha de caducidad ..../.........

☐ Domiciliación bancaria. (Rellenar código cuenta cliente)

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS, S.R.L. que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

CÓDIGO DE CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

- ESTA OFERTA ANULA LAS ANTERIORES
- Oferta válida hasta fin de existencias • Válido para Península y Baleares • Canarias consultar

SÓLO PROGRAMADORES

☑ **Sí, deseo recibir los siguientes números atrasados** ....
....
.... **de SÓLO PROGRAMADORES al precio unitario de 995 ptas. (IVA incluido)**

números 1, 2, 3, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 23, 24 y 26 agotados

Nombre ....Apellidos ....

Domicilio: calle ....Nº ....Piso ....

C.P: ....Población:..Provincia: ....

Teléfono: ....Fax: ....E-mail: ....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº: .... Fecha de caducidad ..../.........

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS, S.R.L. que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

- ESTA OFERTA ANULA LAS ANTERIORES
- Oferta válida hasta fin de existencias • Válido para Península y Baleares • Canarias consultar

Firma:

PEGAR AQUÍ

RESPUESTA COMERCIAL
Autorización nº 14.107
B.O. de C. nº 36
del 18.04.97

NO
NECESITA
SELLO
(A franquear
en destino)

HOJA DE PEDIDO

TOWER
COMMUNICATIONS
APARTADO FD Nº 214
28080 MADRID

Rte.: ....

....

....

suscríbete

a Sólo Programadores

y consigue un **magnífico descuento**

suscripción
normal

ahorro
2.200 ptas.

12 revistas
(1 año)
por sólo...



ptas.

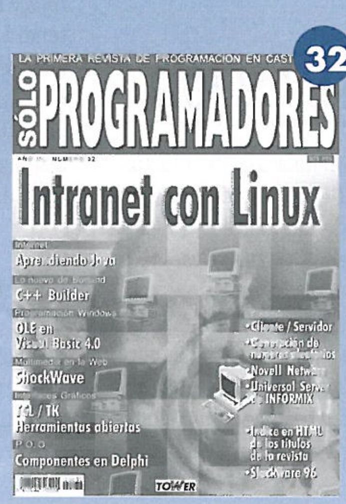
suscripción
estudiantes
(carreras técnicas)

ahorro
4.100 ptas.

12 revistas
(1 año)
por sólo...

7.600 ptas.









Borland C++ Builder 4: la solución integrada

Esteban Amado Buiza (eamadob@hotmail.com)

Borland *C++ Builder 4* es un *RAD (Rapid Application Development)* en el que la palabra "visual" tiene realmente sentido y que permite el desarrollo de todo tipo de aplicaciones en un tiempo récord. Prueba la trial que encontrarás en el CD-ROM.

Con la presentación de la cuarta versión de *C++ Builder*, *Borland* ha demostrado volver a estar al pie del cañón. Para quien no conozca este producto, se puede definir como la herramienta visual de programación en *C++* por excelencia. Ofrece la potencia del *C++*, la productividad de un entorno visual y la experiencia de *Borland*. Además, con las nuevas características añadidas en esta versión se dispone de la posibilidad de utilizar las últimas tecnologías de distribución de aplicaciones que recientemente han aparecido en el mercado informático.

LAS DISTINTAS EDICIONES

En el momento en que se escribió este artículo, *Borland* ha anunciado que va a presentar inicialmente dos versiones de este producto, *Professional* y *Enterprise*. Dentro de unas semanas lanzará al mercado la edición *Standard*. La diferencia que podemos apreciar entre las ediciones *Enterprise* y *Professional* es que esta segunda no posee todas las herramientas que facilitan el desarrollo de objetos distribuidos *CORBA*, ni el soporte *MIDAS* 2 (*Multi-tiered Distributed Aplication Services* ó Servicios de Aplicaciones Multihilo Distribuidas).

LA INSTALACIÓN

El proceso de instalación se efectúa mediante la aplicación *Setup Launcher*, desde donde podremos instalar *C++ Builder 4*, *InterBase 5.5*, *OLEnterprise*, *Remote Debug Server* o el programa especializado en crear instalaciones *InstallShield Express* (edición personalizada para *Borland*). A la hora de instalar *C++ Builder* podemos elegir entre cuatro tipos de instalaciones, teniendo en cuenta que los puntos siguientes se refieren a la edición *Enterprise*:

INSTALACIÓN MÍNIMA

La instalación mínima de *C++Builder* incluye *Borland C++Builder* y *DBE*. Esta instalación ocupará cerca de 175 *Mb* de espacio en disco, pero no se incluirán aplicaciones tan útiles como *WinSight32*, *Image Editor* o *SQL Builder*. Tampoco se tendrá acceso a los fuentes de la *VCL*, a los ejemplos proporcionados por *Borland*, entre otras restricciones, ni para *OWL* ni *MFC*, además, tampoco se dispondrá de los ficheros de ayuda del *SDK* de *Microsoft*.

INSTALACIÓN TÍPICA

Incluye la mayoría de los componentes necesarios para el desarrollo de aplicaciones. Utiliza alrededor de 210 *Mb* de espacio en disco, pero no inclu-

ye soporte ni para *OWL* ni para *MFC*, además, tampoco se instalarán los ficheros de ayuda del *SDK* de *Microsoft*.

INSTALACIÓN COMPLETA

La instalación completa incluye todas las herramientas disponibles así como el soporte necesario para cualquier desarrollador de alguna aplicación basada en una plataforma *Windows* de 32 *bits* (*W95*, *W98* y *Windows NT*). Requiere aproximadamente 300 *Mb* de espacio libre en el disco.

INSTALACIÓN PERSONALIZADA

La instalación personalizada permite determinar exactamente qué componentes quieren instalarse además de permitir añadir componentes adicionales cuando ya se ha realizado una instalación anterior. Esta opción presenta la pantalla de la instalación completa para permitir eliminar los componentes que se deseen.

REQUERIMIENTOS MÍNIMOS

Para instalar y poder ejecutar *Borland C++ Builder 4* el sistema debe cumplir como mínimo con los siguientes requisitos:

- Poseer un procesador de tipo *Intel Pentium* 90 *MHz* o superior, (para la edición *Professional* se requiere un *486 DX2* a 100 *MHz* o superior).
- *Microsoft Windows 95/98 o Windows NT 4.0* (con el *Service Pack 3*)
- Al menos 175*Mb* de espacio libre en disco, aunque se recomienda poseer 300 *Mb*.
- 32 *Mb* de memoria *RAM* como mínimo, si bien 64 *Mb* es lo más recomendable.
- Un ratón o cualquier dispositivo apuntador para *Windows*.
- Una tarjeta de vídeo *VGA* o superior.
- Un lector de *CD-ROM*.

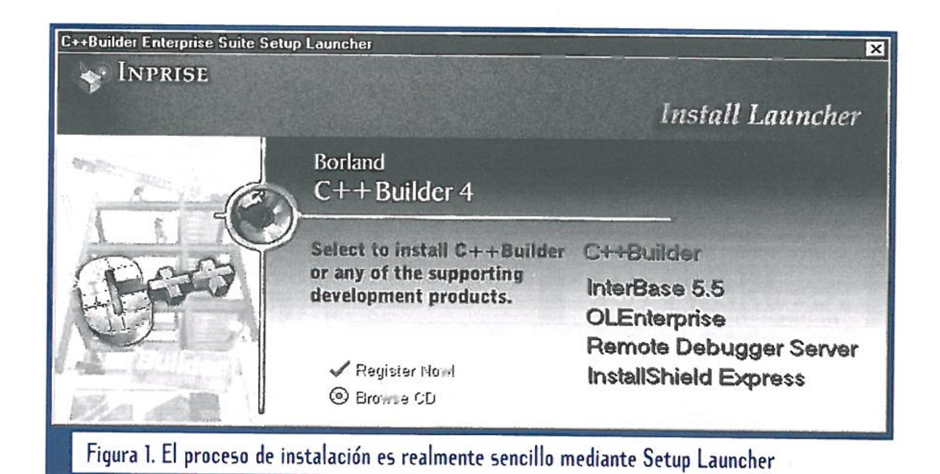


Figura 1. El proceso de instalación es realmente sencillo mediante Setup Launcher

EL ENTORNO DE DESARROLLO

El *IDDE (Integrated Development and Debugging Environment)* que ofrece *C++ Builder 4* es muy parecido al que utilizaba en las versiones anteriores de este mismo producto, o al que presenta *Delphi 4*, lo que proporciona una gran ventaja para los programadores que están acostumbrados a utilizarlo. En él se pueden diferenciar las siguientes partes:

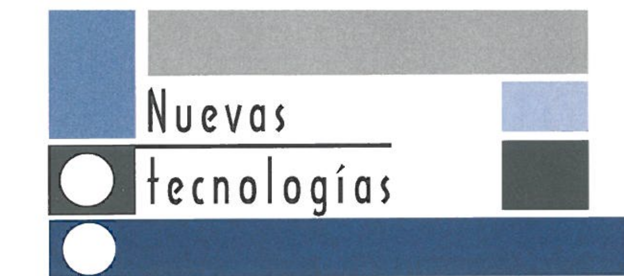
C++ Builder 4 también ha pensado en el año 2000

1. Menú principal y paleta de componentes. El menú principal es el cuadro de mandos del entorno, desde donde se podrá acceder a cualquiera de las opciones que ofrece el producto. La paleta de componentes permite seleccionarlos para incrustarlos.
2. Ventana de gestor de proyecto. Se trata de una herramienta que ayuda a la gestión de los distintos fuentes que forman parte de un proyecto. No está limitado para trabajar con un único proyecto, sino que permite la gestión de varios a la vez, permitiendo activar o desactivar cada uno.
3. Ventana inspector de objetos. Desde esta ventana podremos acceder a las propiedades de los componentes que insertamos en un formulario para modificar el aspecto que proporcionan en tiempo de diseño.
4. Editor de código. Ventana desde la cual se edita el fichero fuente para ir introduciendo el código correspondiente a cada uno de los métodos que hayamos especificado.

LA PALETA DE COMPONENTES

Un componente es un objeto con representación visual que se puede manipular en tiempo de diseño. A su vez, la paleta de componentes es un conjunto de solapas, en cada una de las cuales y organizados por temas, se incluyen los componentes que forman parte de la *VCL* ó Librería de componentes Visuales. Ésta es la verdadera apuesta de *Borland*, con su *VCL* proporciona una serie de componentes que encapsulan las llamadas a la *API* de *Windows*, evitando así que los programadores tengan que volver a inventar la rueda.

Para añadir un componente al formulario que se esté diseñando bastará con hacer doble clic sobre el componente seleccionado o una vez seleccionado, definir en el formulario el área y el tamaño que queremos que ocupe. Veamos a continuación qué tipos de



DirectX 6.1 (II)

Constantino Sánchez Ballesteros (constantino@nexo.es)

Vamos a continuar practicando con *DirectX*, para ello en esta ocasión utilizaremos la *API DirectSound* para efectuar captura de sonido a través de un micrófono instalado en el ordenador.

DIRECTSOUND

DirectSound implementa un nuevo modelo para reproducir y capturar *samples* de sonido digital (incluyendo mezclado de canales). Al igual que los demás elementos de la *API DirectX*, *DirectSound* se aprovecha del *hardware* para obtener el mejor rendimiento posible, y emula características *hardware* mediante software cuando una característica no esté presente.

La reproducción de *DirectSound* está construida sobre la interfaz *COM IDirectSound* y sobre otras interfaces para manipular *buffers* de sonido y efectos *3D*. Estas interfaces son *IDirectSoundBuffer*, *IDirectSound3Dbuffer* e *IDirectSound3DListener*. La captura mediante *DirectSound* se basa en las interfaces *IDirectSoundCapture* e *IDirectSoundCaptureBuffer*. Otra interface *COM*, *IksPropertySet*, provee métodos que permiten a las aplicaciones aprovecharse de las características extendidas de las tarjetas de sonido. Finalmente, *IDirectSoundNotify* es utilizada para señalar eventos cuando la reproducción o captura ha alcanzado un cierto punto en un *buffer*.

INTEGRACIÓN DEL SISTEMA

La siguiente ilustración muestra la relación entre *DirectSound* y los demás componentes del sistema de audio: *DirectSound* y las funciones estándar de *Windows* permiten alternativas a la porción de audio del *hardware* de sonido. Un simple dispositivo permite el acceso de una alternativa en un momento determinado. Si el *driver* de audio ha localizado un dispositivo, un intento por parte de *DirectSound* para localizar un dispositivo devolverá un mensaje de error. De forma similar, si un *driver DirectSound* ha localizado un dispositivo, un intento de localizar el dispositivo por parte del driver de *audio* retornará error.

Si dos dispositivos de sonido están instalados en el sistema, nuestra aplicación puede acceder a cada dispositivo de forma independiente a través de *DirectSound* o las funciones de sonido de *Windows*.

Nota: *Microsoft Video for Windows* actualmente utiliza las funciones de audio de *Windows* para reproducir las pistas de audio de un archivo *AVI*. De este modo, si nuestra aplicación está utilizando *DirectSound* y reproducimos un archivo de este tipo la pista de audio no será audible.

DirectSound aprovecha el hardware para obtener el mejor rendimiento posible

Las aplicaciones pueden liberar el objeto *DirectSound* llamando al método *Release* antes de reproducir un *AVI*.

ABSTRACCIÓN HARDWARE Y EMULACIÓN

DirectSound accede al *hardware* de sonido a través del *HAL*, una interface que es implementada por el *driver* de sonido. *DirectSound HAL* permite las siguientes funcionalidades:

- Adquiere y libera el control del *hardware* de audio.
- Describe las características del *hardware* de audio.
- Crea la operación especificada cuando el *hardware* está disponible.
- Devuelve errores cuando el *hardware* no está disponible.

El *driver* de dispositivo no crea ninguna emulación *software*; simplemente devuelve las características del *hardware* a *DirectSound* y pasa las necesidades de *DirectSound* al *hardware*. Si este último no puede ejecutar una operación determinada, el *driver* del dispositivo devuelve un fallo y *DirectSound* emula dicha operación.

Nuestra aplicación puede utilizar *DirectSound* siempre que se tengan instalados en el equipo los archivos *RunTime* de *DirectX*. Si el *hardware* de sonido no tiene instalado un *driver* para *DirectSound*, éste utilizará emulación *(HEL)*, empleando las funciones de *Windows* (*waveIn* y *waveOut*).

DirectSound soporta captura de samples digitales mediante DirectSoundCapture

La mayoría de las características de *DirectSound* disponibles a través de *HEL* no utilizarán aceleración *hardware*. Para saber cuándo una aplicación está corriendo bajo un *driver* de emulación, utilizaremos el método *GetCaps* y chequearemos el *flag DSCAPS _EMULDRIVER*.

DirectSound se aprovecha automáticamente del *hardware* de sonido acelerado, incluyendo *mixing* y *buffers* de sonido en memoria. Nuestro programa no necesita preguntar para hacer uso de esta aceleración. De todos modos, para efectuar el mejor uso posible de los recursos *hardware* podemos preguntar a *DirectSound* en tiempo de ejecución (utilizando los diversos métodos de *GetCaps*) para recibir una descripción total de las características del dispositivo de sonido y utilizar diferentes rutinas optimizadas para la presencia o ausencia de una característica determinada. También podemos especificar qué *buffers* de sonido deberían utilizar aceleración *hardware*.

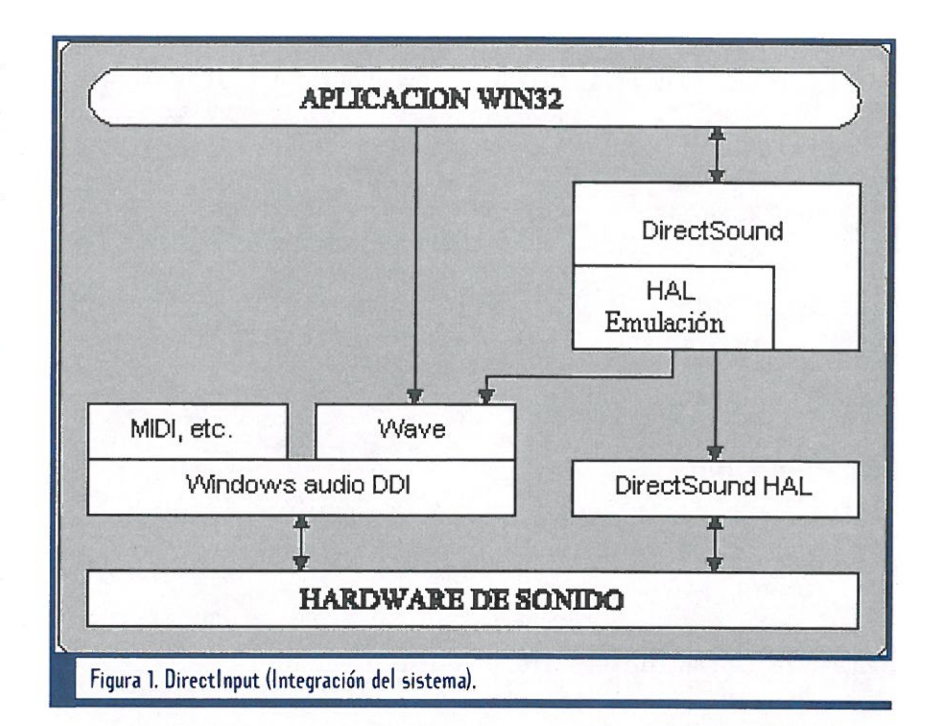


Figura 1. DirectInput (Integración del sistema).

DATOS DE SONIDO

DirectSound y *DirectSoundCapture* trabajan con datos de audio en formato de onda que consisten en *samples* digitales de sonido a una frecuencia determinada. El formato particular de un sonido puede ser descrito por la estructura *WAVEFORMATEX* tal y como se muestra a continuación:

```
typedef struct {
    WORD  wFormatTag;
    WORD  nChannels;
    DWORD nSamplesPerSec;
    DWORD nAvgBytesPerSec;
    WORD  nBlockAlign;
    WORD  wBitsPerSample;
    WORD  cbSize;
} WAVEFORMATEX;
```

El miembro *wFormatTag* contiene un identificador único asignado por *Microsoft*. La única etiqueta válida con *DirectSound* es *WAVE_FORMAT _PCM*, que indica *Pulse Code Modulation (PCM)*, un formato no comprimido en el cual cada *sample* representa la amplitud de la señal en tiempo de muestreo. *DirectSoundCapture* puede capturar datos en otros formatos utilizando el *Manager* de compresión de audio.

El miembro *nChannels* describe el número de canales, normalmente uno (mono) o dos (estéreo). Para datos en estéreo, los *samples* son entrelazados. El miembro *nSamplesPerSec* describe el ratio de muestreo, o frecuencia, en *Hertzios*. Los valores típicos son 11.025, 22.050, y 44.100. El miembro *wBitsPerSample* devuleve el tamaño de cada *sample*, generalmente 8 ó 16 *bits*. El valor en *nBlockAlign* es el número de *bytes* requerido para cada *sample* completo, y para formatos *PCM* es igual a (**wBitsPerSample * nChannels / 8**). El valor en *nAvgBytesPerSec* es el producto de *nBlockAlign* y *nSamplesPerSec*.

Finalmente, *cbSize* devuleve el tamaño de cualquier campo extra

```
&mmckinfoData, &mmckinfoParent))
        return FALSE;

    if
    (WaveStartDataWrite(&hmmio,
    &mmckinfoData, &mmioinfo)) {

    WaveCloseWriteFile(&hmmio,
    &mmckinfoData,
&mmckinfoParent, &mmioinfo,
dwTotalBytesWritten / (wfx.wBits-
    PerSample / 8));
DeleteFile(pszFileName);
return FALSE;
    }
    if FAILED(IDirectSoundCaptu-
    reBuffer_Start(lpdscb,
DSCBSTART_LOOPING)) {

    WaveCloseWriteFile(&hmmio,
    &mmckinfoData,
&mmckinfoParent, &mmioinfo, 0);
        DeleteFile(pszFileName);
        return FALSE;
    }

    dwTotalBytesWritten = 0;

    return TRUE;
}
```

En primer lugar esta función llama a la función *WaveCreateFile* establecida en el archivo del *SDK* **WAVE.C**, para poder crear un fichero *RIFF* y escribir la cabecera del formato *WAV*. Seguidamente se llama a la función *WaveStartDataWrite* la cual avanza el puntero del fichero al *chunk* de datos. Finalmente, comienza la captura en el *buffer*. En medio segundo nuestra aplicación notificará que los datos son válidos y debemos prepararnos para copiarlos al archivo.

Debemos tener en cuenta la inicialización de *dwTotalBytesWritten*. Este valor va a ser necesitado para la cabecera del *chunk* de datos después de que se complete la captura.

- **Paso 6: Gestionando las notificaciones de captura.** Nosotros recibimos notificaciones de captura como eventos en el bucle de mensajes, al igual que las notificaciones de reproducción. A continuación se muestra un ejemplo de este tipo de bucles:

```
BOOL Done = FALSE;
while (!Done) {
    DWORD dwEvt = MsgWaitForMul-
    tipleObjects(
            NUMCAPTUREEVENTS,
    // Número de posibles eventos
            rghEvent,
    // Localización de handles
            FALSE,
    // Esperamos?
            INFINITE,
    // Tiempo de espera
            QS_ALLINPUT);
    // cualquier mensaje es un
    evento

    dwEvt -= WAIT_OBJECT_0;

    // si el evento fue asignado
    por el buffer, existe entrada
    para procesar

    if (dwEvt < NUMCAPTUREEVENTS)
    {
        StreamToFile();
    }

    // si es el último mensaje,
    también lo trataremos como
    mensaje
    else if (dwEvt == NUMCAPTURE-
    EVENTS) {
        while (PeekMessage(&msg,
    NULL, 0, 0, PM_REMOVE)) {
            if (msg.message ==
    WM_QUIT) {
                Done = TRUE;
            }
            else {
                TranslateMessa-
    ge(&msg);
                DispatchMessa-
    ge(&msg);
            }
        }
    }  // Fin del procesamiento
    de mensajes
}
```

Si estamos recibiendo notificaciones del *buffer* de captura y el *buffer* secundario, necesitaremos distinguir entre los dos tipos de eventos por el valor del índice en *dwEvt*.

Por ejemplo, los eventos 0 y 1 podrían ser notificaciones, y los eventos 2 y 3 podrían ser notificaciones de captura.

Los datos de sonido capturados se pueden guardar a un archivo con formato WAV

- **Paso 7: Envío de datos al archivo WAV.** Para enviar datos de sonido capturados a un archivo *WAV* utilizaremos la función *StreamToFile*:

```
BOOL StreamToFile(void) {
  DWORD          dwReadPos;
  DWORD          dwNumBytes;
  LPBYTE     pbInput1, pbInput2;
  DWORD      cbInput1, cbInput2;
  static DWORD  dwMyReadCursor=0;
  UINT        dwBytesWritten;
```

Atención a la declaración estática de *dwMyReadCursor*. Este es el desplazamiento del próximo *byte* de datos que queremos leer; en otras palabas, el *byte* justo por debajo del último leído en el paso previo a esta función. La primera acción que hace la función es encontrar la posición de lectura actual.

Se debe recordar que esta posición marca la esquina principal del dato que es salvado para guardar. No es necesario el mismo que el de la notificación de posición dado que éste está más avanzado desde que se señaló el último evento.

```
    IDirectSoundCaptureBuffer_Get
    CurrentPosition(lpdscb,
                NULL, &dwReadPos);
```

La función sustrae nuestra posición de lectura interna a partir de la posición de lectura actual para poder determinar cuántos *bytes* de nuevos datos están disponibles.

```
if (dwReadPos < dwMyReadCursor)
dwReadPos += dscbDesc.
     dwBufferBytes;
dwNumBytes = dwReadPos - dwMyRe-
     adCursor;
```

Dado que el segmento de datos que hemos identificado como disponible no está exactamente delimitado por las notificaciones de posiciones al comienzo y mitad del *buffer*, la porción protegida del *buffer* podría "envolverse", en caso de que se requieran dos operaciones:

```
if FAILED(IDirectSoundCaptureBuf-
     fer_Lock(lpdscb,
dwMyReadCursor, dwNumBytes,
(LPVOID *)&pbInput1, &cbInput1,
(LPVOID *)&pbInput2, &cbInput2, 0))
OutputDebugString("Fallo en pro-
     tección de captura ");
else {
if (WaveWriteFile(hmmio, cbInput1,
     pbInput1, &mmckinfoData,
&dwBytesWritten, &mmioinfo))

     OutputDebugString("Fallo en
     la escritura de datos al
     fichero\n");
dwTotalBytesWritten += dwBytesW-
     ritten;

// Envolvemos
if (pbInput2 != NULL) {
if (WaveWriteFile(hmmio, cbInput2,
     pbInput2,
&mmckinfoData, &dwBytesWritten,
     &mmioinfo))
OutputDebugString("Fallo en la
     escritura de datos al fiche-
     ro\n");
dwTotalBytesWritten + = dwBytes
     Written;
}

IDirectSoundCaptureBuffer_Unlock
     (lpdscb,
pbInput1, cbInput1,
pbInput2, cbInput2);
}
```

La función *WaveWriteFile* devuelve 0 si todo se realizó correctamente y rellena *dwBytesWritten* con el número de *bytes* actualmente copiados al archivo.

Podemos establecer notificaciones en los buffers para saber qué ocurre mientras se efectúa una captura

Finalmente, actualizamos la posición interna de lectura:

```
dwMyReadCursor += dwNumBytes;
     if (dwMyReadCursor >= dscb-
     Desc.dwBufferBytes)
          dwMyReadCursor -=
     dscbDesc.dwBufferBytes;

     return TRUE;
} // end StreamToFile()
```

- **Paso 8: Parada de la captura.** Cuando sea el momento de parar la grabación, llamaremos a la siguiente función:

```
BOOL StopWrite() {

     IDirectSoundCaptureBuffer_Sto
     p(lpdscb);
StreamToFile();
WaveCloseWriteFile(&hmmio, &mmc-
     kinfoData,
&mmckinfoParent, &mmioinfo,
dwTotalBytesWritten / (wfx.wBits-
     PerSample / 8));
return TRUE;
}
```

Esta función para el *buffer* de captura, llama a la función *StreamToFile* una vez más para grabar todos los datos hasta la posición de lectura y cierra el archivo. La función *WaveCloseWriteFile* del archivo del *SDK WAVE.C* actualiza la cabecera del *chunk* de datos escribiendo el número total de *samples*.

- **Paso 9: Liberando el objeto DirectSoundCapture.** Antes de cerrar nuestra aplicación debemos liberar el sistema de captura. Es necesario liberar la interfaz *IDirectSoundNotify* antes de hacer lo mismo con el *buffer* de captura.

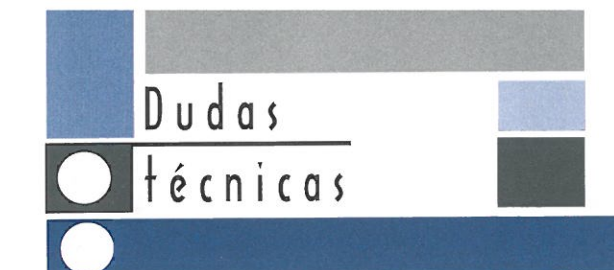
```
void CleanupDSoundCapture(void) {
if (lpdsNotify)
IDirectSoundNotify_Release
     (lpdsNotify);

if (lpdscb)
IDirectSoundCaptureBuffer_
     Release(lpdscb);
if (lpdsc)
          IDirectSoundCapture_
     Release(lpdsc);
}
```

IMPORTANCIA DE LOS FORMATOS DE BUFFERS

El mezclador de *DirectSound* convierte los datos de cada *buffer* secundario en el formato establecido para el *buffer* primario. Esta conversión se realiza en tiempo real cuando se mezclan los datos en el *buffer* primario y, consecuentemente, baja el rendimiento de la aplicación al utilizar numerosos ciclos de reloj de la *CPU* para efectuar esta tarea. Podemos eliminar esta sobrecarga asegurándonos que los *buffers* secundarios y el primario tienen el mismo formato.

Dada la forma en que *DirectSound* efectúa la conversión de formato, sólo necesitamos establecer el mismo *ratio* y número de canales. No importa si existe una diferencia en el tamaño del *sample* (8 *bits* o 16 *bits*).



Dudas técnicas

En esta sección, como cada mes, SÓLO PROGRAMADORES os brinda la oportunidad de encontrar respuesta a las dudas que podáis tener en algún tema relacionado con la programación bajo cualquier entorno de desarrollo.

Pregunta

Soy una asidua lectora de vuestra revista, la cual encuentro muy interesante. Últimamente he podido leer muchos artículos relacionados con bases de datos, tema en el cual me encuentro muy cómoda y en el que estoy comenzando a realizar aplicaciones serias. Estando acostumbrada al mundo *Unix*, no encuentro interesantes sistemas de acceso como pudieran ser *OLE*, *COM* u *ODBC*. Sin embargo, encuentro muy interesantes herramientas como el *Pro*C*, una herramienta de *ORACLE* diseñada para acceder a los datos del sistema a través de programas en lenguaje *C*.

He comenzado a desarrollar algunos programas con bastante éxito. Sin embargo, he encontrado un problema de difícil solución: todos los accesos se realizan a tablas conocidas de antemano, pero existen casos en los que podemos querer ser más generales, utilizando una única rutina para acceder a diferentes tipos de tablas. ¿Se puede realizar este tipo de operaciones desde *Pro*C*?, ¿Cómo podríamos insertar nuevos datos en una tabla, independientemente de su naturaleza? Os doy las gracias por adelantado, aunque sería estupendo que ofrecierais un curso de *Pro*C* a través de la revista.

Respuesta

Estimada lectora:

Quiero comentarte que la respuesta a la primera pregunta que formulas es muy fácil. Efectivamente, esta herramienta - *el Pro*C* - sería bastante inútil si no pudiera utilizarse de una manera totalmente genérica.

De hecho, una gran parte de los productos de acceso a *ORACLE* que ofrece la compañía, están realizados mediante este tipo de entornos de desarrollo.

Por lo tanto, podemos responder afirmativamente a tu pregunta. Si que es posible acceder a tablas de las cuales desconocemos la estructura en tiempo de generación del código. Para ello hemos de utilizar alguno de los cuatro métodos dinámicos que ofrece *Pro*C*.

En cuanto a la selección del método adecuado, todo dependerá de las características particulares de nuestro problema. A continuación describimos los diferentes métodos, así como las condiciones en las que deberemos utilizar cada uno de ellos.

- Método 1: es el más limitado. Sólo podemos insertar datos o realizar operaciones que no devuelvan ningún tipo de resultado. Además, no podemos utilizar las denominadas variables *host* al construir *queries* con él.

- Método 2: una evolución sobre el anterior, nos permite utilizar variables *host* en la definición de los parámetros de entrada. Por lo tanto, aunque es más flexible que el método 1, no nos permite recuperar datos que se encuentren almacenados en *ORACLE.*

- Método 3: es el primero de los sistemas utilizados para recuperar información de una manera dinámica. Con este sistema podemos definir cursores basados en *queries* dinámicas. Sin embargo, este método presenta una serie de limitaciones, principalmente en el caso de que el número de variables *host* a utilizar sea desconocido en tiempo de compilación.

- Método 4: es el más complejo y a su vez el más completo. Con este

sistema tenemos un acceso totalmente arbitrario a la base de datos. Su desventaja principal radica en la necesidad de que el programador conozca de una manera muy precisa el mecanismo de comunicación con el *RDBMS*. Además, el desarrollador ha de estar íntimamente familiarizado con las estructuras de datos que posibilitan la transferencia de datos.

De todos los anteriores, el método cuatro es el más flexible pero también altamente desaconsejable por su complejidad inherente. Los tres primeros sirven en el 99% de los casos si el diseño del programa es adecuado.

En cuanto a la segunda pregunta, si lo que quieres es insertar datos de cualquier tipo en la base de datos, podemos utilizar el método 1 ó el método 2. En la mayoría de los casos, el primero de ellos será suficiente, siendo necesario recurrir al segundo cuando tengamos datos binarios o de gran tamaño.

El siguiente ejemplo supone que la conexión a la base de datos se ha efectuado. En él vamos a introducir un número arbitrario de números enteros en una tabla. Los nombres de los campos de la tabla se encuentran en el *array* **camposTabla**, mientras que los números están almacenados en **valoresFinales**. La variable **numeroValores** indica cuántas columnas existen.

```
#define  ERROR   1
#define     EXITO      0
Char   camposTabla[50][50];
int    valoresFinales[50];

...

Int
insercionDinamica ( nombreTa-
bla, camposTabla , valoresFi-
nales, numeroValores )
 char * nombreTabla;
 char * camposTabla[];
 int * valoresFinales;
 int numeroValores;
{
EXEC SQL BEGIN DECLARE SEC-
TION;
 VARCHAR       COMANDO[2000];

EXEC SQL END DECLARE SECTION;

int i;
char tmpstr[100];


if (  (numeroValores > 50 )
||
       (numeroValores < 1 ) )
{
 printf ("Demasiados
campos.\n");
 return ERROR;
}

sprintf ( COMANDO.arr ,
 "INSERT INTO     %s ( ",
 nombreTabla );

for ( i = 0 ; i <
numeroValores ; i++ )
 {
 if ( i  != 0 )  strcat (
COMANDO.arr , " ," );

 strcat ( COMANDO.arr ,
camposTabla[i] );
 }

strcat ( COMANDO.arr , " )
\n VALUES (  " );

for ( i = 0 ; <
numeroValores ; i++ )
 {
 if ( i  != 0 )  strcat (
COMANDO.arr , " ," );

 sprintf ( tmpstr , " %d",
valoresFinales[i] );
 }

strcat ( COMANDO.arr , " )"
);

COMANDO.len = strlen (
COMANDO.arr );
EXEC SQL EXECUTE IMMEDIATE
:COMANDO;

if ( sqlca.sqlcode != 0 )
 return ERROR;
return EXITO;

}
```

Como podemos ver en el ejemplo, la idea es construir la query como si estuviéramos en el entorno interactivo, para luego enviarla a la base de datos con el comando **EXECUTE**.

En la función generamos dinámicamente los campos uno a uno, para a continuación poder generar los valores. Finalmente, capturamos el código de error para verificar el resultado de la *query*.

Pregunta

Amigos de Sólo Programadores:

Hace ya varios años que programo en el entorno de *Windows*, tanto en las antiguas versiones (3.1 y 3.11) como en las más recientes *Windows95 y NT*.

Tras haber leído y escuchado incluso en las noticias la potencia y estabilidad de los sistemas *Unix*, he decidido probar suerte en este entorno.

Para ello, he instalado *Linux* en casa y he conseguido acceso a una *UltraSparc* de *Sun* en el trabajo. Creo que he conseguido configurarlas correctamente, - al menos el PC de casa, ya que la estación tiene un administrador, - pero no sé como empezar a programar. Sé que ambos tienen instalado el *GCC*, ¿significa esto que los programas de un lado se pueden compilar en el otro y funcionan?

Por otro lado, estoy editando con un programa que se denomina *VI*, el cual no es precisamente muy cómodo

para programar. ¿Existe algún entorno integrado como los de *Windows*?

Finalmente, ¿cómo se depuran los progamas? y ¿cómo puedo saber qué llamadas al sistema tengo a mi disposición?

Muchas gracias por adelantado.

Respuesta

El cambio del entorno *Windows* a un sistema *Unix*, requiere cambiar la filosofía general de trabajo. Así como en *Windows* encontrarás programas autocontenidos, - como el compilador de *C*, - los cuales pretenden satisfacer todas tus necesidades, en *Unix* normalmente encuentras diversas herramientas para cada tarea, de tal manera que el usuario final decida cuales son las que más le gustan.

En cuanto a tu primera pregunta, la respuesta es: casi siempre. Normalmente, basta recompilar en la *Sun* tus programas *Linux* o viceversa para que puedas disfrutar de ellos.

En cualquier caso, a la hora de compilarlos, comprueba los errores y los *warnings*, ya que es posible que algunas funciones no sean exactamente iguales.

Sin embargo, utilizando las páginas de manual, es posible encontrar las diferencias y preparar tu programa para ellas utilizando directivas del preprocesador.

```
#ifdef LINUX

 ... código Linux ...

#endif
#ifdef SOLARIS

 ... código Sun ...

#endif
```

La elección de un editor de desarrollo suele ser un tema religioso entre los usuarios de *Unix*. El propio *VI*, es uno de los más potentes que existen, aunque su curva de aprendizaje es increíblemente empinada.

En cualquier caso, *EMACS* o *XEMACS*, así como *NEDIT* satisfarán con creces tus necesidades. Todos ellos están disponibles de manera gratuita en *Internet*.

Entornos integrados de desarrollo existen varios, aunque entre los gratuitos destaca *XWindows Programming Environment*, también llamado *XWPE*. Está disponible para *Linux* y se incluye en *RedHat 5.2*. Sin embargo, en nuestra opinión, nada es comparable a utilizar la combinación *GCC/EMACS/GDB/MAN* como compilador/editor/depurador/ documentación.

El depurador incorporado en el sistema es el famoso *GDB*, aunque existen algunos más cómodos de utilizar, entre los que destaca el *Data Display Debugger (DDD)*.

Finalmente, las llamadas al sistema se documentan en la sección 2 de las páginas del manual. Podrás acceder a ellas una a una desde línea de comando, o a todas con el programa *Xman*.

Pregunta

Lo primero que quiero es felicitaros por la revista y la cantidad de trucos y formas de conseguir que los ordenadores sean más amigables que nos facilitáis. Lo segundo es consultar un asunto referente al curso de programación de *Glide*.

Hace poco me compré una aceleradora *Voodoo 2*, con la idea de utilizar lo que estaba aprendiendo en los artículos de la revista. Instalé el *SDK* que acompañaba al CD de la revista nº 52 para seguir el curso, utilizando *Visual C++ 5.0*.

Seguí los pasos indicados en el artículo y compilé el primer ejemplo que venía en él. Me sorprendió mucho ver que no reconocía el tipo de la única variable que se creaba en el programa.

Considerando que me había equivocado al teclear el tipo, ya que las funciones las reconocía, lo que implica que los paths a las librerías estaban bien, copié el código fuente del CD y lo compilé, observando que se producía el mismo error en el mismo punto.

Mis preguntas son, ¿los artículos se han realizado con *Glide 2*? Si es así, ¿habéis comprobado que las versiones 2 y 3 son totalmente compatibles? Gracias por todo.

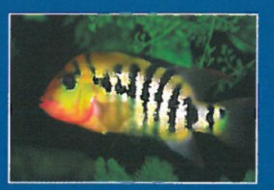
Respuesta

Estimado lector en respuesta a tu pregunta te comentamos que la versión que se está utilizando en la revista sobre el curso de *Glide* es la 2.43. Ésta es la versión del *SDK*, no de los *Runtimes*.

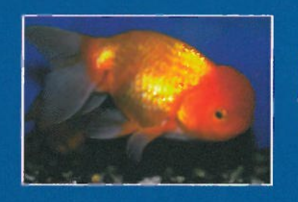
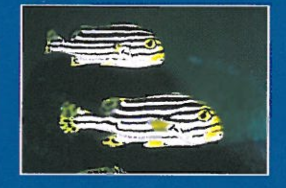
Aunque se utilice esta versión para programar, no hay ningún problema en tener instalados los *drivers Runtime* de la versión 3 de *Glide* ya que es compatible con otras versiones.

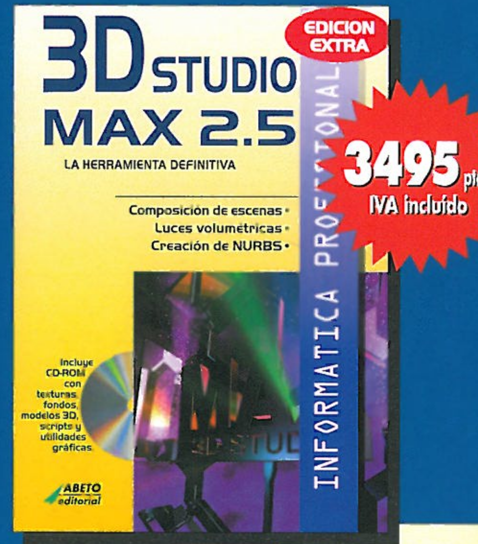
Por supuesto, si se quiere utilizar esta nueva versión del *SDK* (la 3.X) será necesario retocar el código cambiando a los nuevos tipos de variables y posibles nuevos miembros implementados en las funciones.

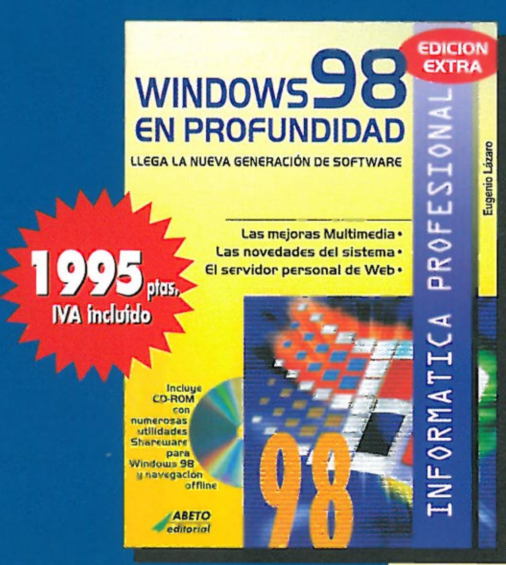
De todos modos, la esencia de las rutinas creadas en el curso no cambia para nada tanto si se utiliza la versión 2.X o 3.X de *Glide*.

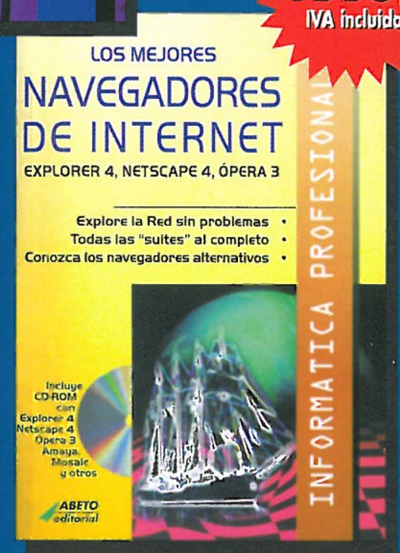
PARA  MOVERTE  EN 

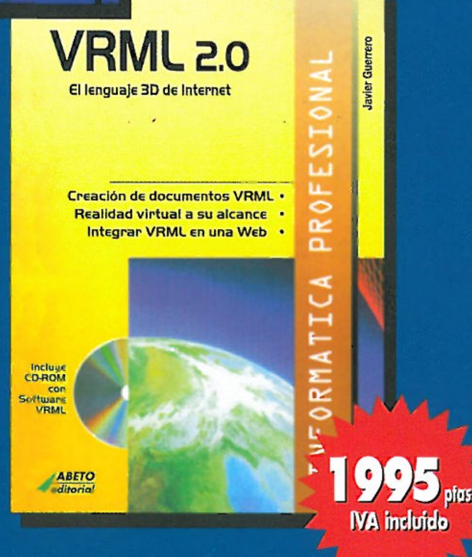
INFORMÁTICA  COMO  PEZ

 EN  EL  AGUA 









WINDOWS 98 ¡YA!
Cómo disponer de Windows 98 con Windows 95 y este libro...
Javier Guerrero
Documentos locales y remotos •
El escritorio activo •
La nueva interfaz •
Incluye CD-ROM con todo lo necesario
INFORMATICA PROFESIONAL
ABETO editorial
1995 ptas. IVA incluido

Incluyen CD-ROM de regalo

Ya a la venta en quioscos y librerías.

ABETO editorial

c/ Aragoneses, 7 • 28108 Alcobendas (Madrid)
Tel.: 91 661 42 11* • Fax: 91 661 43 86